

NASA/TM-1998-208613

SOFTWARE ENGINEERING LABORATORY SERIES

SEL-96-001

IN-87

415440

31.

COLLECTED SOFTWARE ENGINEERING PAPERS: VOLUME XIV

OCTOBER 1996



National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

Page intentionally left blank

FOREWORD

The Software Engineering Laboratory (SEL) is an organization sponsored by the National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC) and created to investigate the effectiveness of software engineering technologies when applied to the development of application software. The SEL was created in 1976 and has three primary organizational members:

NASA/GSFC, Flight Dynamics Systems Branch

University of Maryland, Department of Computer Science

Computer Sciences Corporation, Software Engineering Organization

The goals of the SEL are (1) to understand the software development process in the GSFC environment; (2) to measure the effect of various methodologies, tools, and models on this process; and (3) to identify and then to apply successful development practices. The activities, findings, and recommendations of the SEL are recorded in the Software Engineering Laboratory Series, a continuing series of reports that includes this document.

Single copies of this document can be obtained by writing to

Flight Dynamics Systems Branch

Code 551

Goddard Space Flight Center

Greenbelt, Maryland, U.S.A. 20771

Page intentionally left blank

TABLE OF CONTENTS

0—	Section 1—Introduction	1
0—	Section 2—Software Models	3
1—	"Software Engineering Technology Infusion Within NASA," M. Zelkowitz	5
2—	<i>Communication and Organization in Software Development: An Empirical Study,</i> V. Basili, C. B. Seaman.....	17
3—	"Understanding and Predicting the Process of Software Maintenance Releases," V. Basili, L. Briand, S. Condon, W. Melo, J. Valett.....	53
4—	"The Role of Experimentation in Software Engineering: Past, Current, and Future," V. Basili.....	65
5—	"Simulation Modeling of Software Development Processes," G. Calavaro, V. Basili, G. Iazeolla	73
	Section 3—Technology Evaluations	79
6—	<i>Qualitative Analysis for Maintenance Process Assessment,</i> L. Briand, , Y. Kim	81
7—	"Evolving and Packaging Reading Technologies," V. Basili.....	117
8—	<i>The Empirical Investigation of Perspective-Based Reading,</i> V. Basili, S. Green, O. Laitenberger, F. Shull, S. Sorumgaard, M. Zelkowitz	129
9—	"The Impact of Ada and Object-Oriented Design in NASA Goddard's Flight Dynamics Division," S. Waligora, J. Bailey, M. Stark.....	171
	Standard Bibliography of SEL Literature	191

SECTION 1—INTRODUCTION

This document is a collection of selected technical papers produced by participants in the Software Engineering Laboratory (SEL) from September 1995 through September 1996. The purpose of the document is to make available, in one reference, some results of SEL research that originally appeared in a number of different forums. This is the 14th such volume of technical papers produced by the SEL. Although these papers cover several topics related to software engineering, they do not encompass the entire scope of SEL activities and interests. Additional information about the SEL and its research efforts may be obtained from the sources listed in the bibliography at the end of this document, or via the SEL Home Page on the World Wide Web at <http://fdd.gsfc.nasa.gov/seltext.html>.

For the convenience of this presentation, the nine papers contained here are grouped into two major sections:

- Software Models (Section 2)
- Technology Evaluations (Section 3)

Section 2 includes several papers that describe technology transfer and the technology infusion process, communication issues among members of a software development organization, and a case study to better understand the effort distribution of releases and build a predictive effort model for software maintenance releases. Section 2 also includes papers that discuss the role of experimentation and process improvement in industrial development, and a simulation modeling approach for predicting software process productivity indices. Section 3 includes four papers. The first contains a characterization process aimed specifically at maintenance and based on a general qualitative analysis methodology. The second paper, through a series of experiments, provides a motivation for reading as a quality improvement technology, based upon experiences in the SEL at NASA Goddard; the third discusses Perspective-Based Reading (PBR), a new reading technique for requirements documents; and the fourth paper highlights the key findings of 10 years of use and study of Ada and object-oriented design in NASA Goddard's Flight Dynamics Division (FDD).

The SEL is actively working to understand and improve the software development process at the Goddard Space Flight Center (GSFC). Future efforts will be documented in additional volumes of the *Collected Software Engineering Papers* and other SEL publications.

omit

SECTION 2—SOFTWARE MODELS

The technical papers included in this section were originally prepared as indicated below.

- "Software Engineering Technology Infusion Within NASA," M. Zelkowitz, *IEEE Transactions on Engineering Management*, vol. 43, no. 3, August 1996
- *Communication and Organization in Software Development: An Empirical Study*, V. R. Basili and C. B. Seaman, University of Maryland, Computer Science Technical Report, CS-TR-3619, UMIACS-TR-96-23, April 1996
- "Understanding and Predicting the Process of Software Maintenance Releases," V. R. Basili, L. Briand, S. Condon, W. Melo, J. Valett, *Proceedings of the 18th International Conference on Software Engineering (ICSE-18)*, Berlin, Germany, March 1996
- "The Role of Experimentation in Software Engineering: Past, Current, and Future," V. R. Basili, *Proceedings of the 18th International Conference on Software Engineering (ICSE-18)*, Berlin, Germany, March 1996
- "Simulation Modeling of Software Development Processes," G. F. Calavaro, V. R. Basili, and G. Iazeolla, *7th European Simulation Symposium (ESS '95)*, October 1995

Software Engineering Technology Infusion Within NASA

Marvin V. Zelkowitz, Senior Member, IEEE

51-61

415767

360825

124

Abstract—Abstract technology transfer is of crucial concern to both government and industry today. In this paper, several software engineering technologies used within NASA are studied, and the mechanisms, schedules, and efforts at transferring these technologies are investigated. The goals of this study are: 1) to understand the difference between technology transfer (the adoption of a new method by large segments of an industry) as an industrywide phenomenon and the adoption of a new technology by an individual organization (called technology infusion) and 2) to see if software engineering technology transfer differs from other engineering disciplines. While there is great interest today in developing technology transfer models for industry, it is the technology infusion process that actually causes changes in the current state of the practice.

I. INTRODUCTION

THE ability to move a new technology from a development laboratory into general use in industry is of increasing concern as today's economic climate constantly reduces the time available for companies to develop new products. This process, generally called *technology transfer*, is of crucial concern as industry today needs to remain economically competitive in the global marketplace.

Technology transfer is a difficult and slow process [12, p. 1]

One reason why there is so much interest in the diffusion of innovations is because getting a new idea adopted, even when it has obvious advantages, is often very difficult. There is a wide gap in many fields between what is known and what is actually put into use. Many innovations require a lengthy period, often of some years, from the time when they become available to the time when they are widely adopted. Therefore, a common problem for many individuals and organizations is how to speed up the rate of diffusion of an innovation.

The software development community is aware of this problem and the need to diffuse new innovations, i.e., transfer effective technology toward improving the process of developing software. This is a major goal toward achieving improvements in productivity and reliability of the resulting products. Concepts like the Software Engineering Institute's Capability Maturity Model [10] have grown in importance as a means for modifying the software development process.

Manuscript received February 28, 1995. This work was supported in part by Grant NSG-5123 from NASA Goddard Space Flight Center to the University of Maryland. Reviews of this manuscript were arranged by Department Editor B. V. Dean.

The author is with the Institute for Advanced Computer Studies, and Department of Computer Science, University of Maryland, College Park, MD 20742 USA.

Publisher Item Identifier S 0018-9391(96)05610-3.

The "experience factory" concept of the National Aeronautics and Space Administration (NASA) Goddard Space Flight Center (GSFC) Software Engineering Laboratory (SEL) [4] has shown the value of process improvement.

However, all process improvement involves changes. Some of these may be relatively minor alterations to the current way of doing business (e.g., replacing one compiler or editor by another), while others may require major changes that affect the entire development process (e.g., using Cleanroom software development and eliminating much of the unit testing phase). In order for an organization to continually improve its process, it must be aware of how it operates and what other technologies are available that may be of use.

While much has been written on the general concept of technology transfer within an industry, there is not much which describes the processes which an individual organization undergoes to adopt a new technology. This change generally goes under the name of *technology infusion*. We can describe technology transfer as technology infusion which diffuses across a broad segment of a given industry. In order to investigate these issues, several software engineering technologies that have been adopted by NASA for use on various development projects are studied. In particular, we are interested in the mechanisms that were used to accomplish the infusion of the technology, the effort involved in performing that infusion, and the time that it took to accomplish. This work is part of an in-depth 1993–1994 study of software engineering technology within NASA [16]; however, only those results that seem to be applicable to a more general technological audience are presented in this paper.

In Section II we discuss the general problems of technology transfer and in Section III, we discuss technology infusion at NASA. We show that software engineering seems to follow a technology infusion process that differs from other engineering disciplines. In particular, the lack of a culture in software engineering to experiment and measure results makes validation of new technologies difficult. Also the major "products" of software engineering are processes, which makes the discipline behave more like a scientific than an engineering activity. Established technology transfer models do not address this well. These findings are presented in Section IV.

II. TECHNOLOGY TRANSFER

By *technology transfer* we mean the insertion of a new technology into most organizations that perform similar tasks. The insertion must be such that the new organizations regularly use that technology if the appropriate conditions on its use

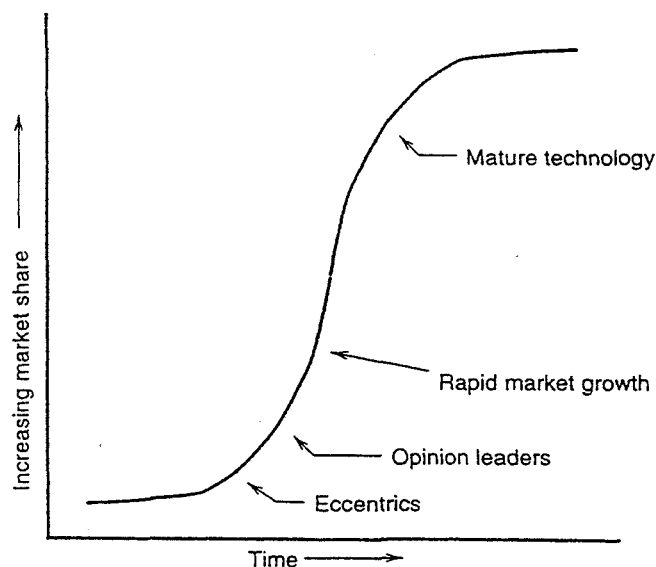


Fig. 1. Product life cycle S-curve growth cycle.

should arise in the future. The organization that adopts the new technology is said to *infuse* that technology. We will call the creator of that technology the *producer* and the organization that accepts and uses the new technology the *consumer* of that technology. The process of moving the technology out of the producing organization will be called *exporting* the technology.

A. Models of Technology Transfer

Technology transfer has typically been identified as an importation process. It often follows the *product life cycle* (PLC) identified by Rogers via the S-curve [12]. The first few customers are the "oddballs" or "eccentrics" of society who adopt a new product. Following them are the "opinion leaders," who then give their approval to the product. Society then follows these opinion leaders, and product growth follows rapidly. During the mature stage, as the market saturates, growth levels off, giving the characteristic S-curve.

1) *Gatekeepers*: Technology transfer follows a similar process. One member of an organization, often called the *gatekeeper* [1], monitors technological developments and chooses those that seem appropriate for inclusion in an organization; hence opens the "gate" to the new technology. Because this role is often informal, it may fall naturally to the most creative and technically astute individual in an organization. Since the gatekeeper is aware of technical developments outside of the organization, others in the group often look toward this person for guidance. This person often is known by the name "guru" or similar sounding monikers.

2) *Transition Models*: However, the gatekeeper is not the only approach toward technology transfer. Other models of the process have been identified. In one study of 44 technology transfer efforts at one aerospace company, Berniker [5] identified four approaches toward technology transfer.

a) *People mover model*: In this approach, there is personal contact between the developer and the user of a tech-

nology. Typically there is some facilitator within the infusing organization that knows about the new technology and wishes to import it into the new organization (i.e., the gatekeeper mentioned above). This method was found to be the most prevalent and effective of all technology transfer methods.

Nochur and Allen [9] investigated this model and discovered that it really consists of three separate subcategories, which we will call:

- 1) the *spontaneous gatekeeper* role assumed by organization member;
- 2) the *assigned gatekeeper* role imposed by management on some organization member;
- 3) the *umbrella gatekeeper* role assumed by another organization to impose new technology on others.

b) *Communication model*: In this approach, the new technology has appeared in print; and as with the people mover model, some facilitator discovers the technology and wishes to infuse it into the new organization. The "print" mechanism may be internal documentation, conference reports, or journal publications.

c) *On-the-shelf model*: This approach, relatively rare among the projects studied by Berniker, requires the new technology to be packaged so that nonexperts can discover it and learn enough about it to begin the infusion process. It requires sufficient documentation so that others can easily pick it up and use it. Reading about the technology in a "parts catalog" is an example of this method.

d) *Vendor model*: This last method requires an organization to turn over the task to a vendor to sell them a new technology. It effectively turns the vendor into the agent of the people mover, communication, or on-the-shelf models.

The communication model and the on-the-shelf model can be viewed as marketplace models. Innovations (in the form of reports, papers, and products) are placed in the marketplace, and users will discover what they need. However, these appear to be very imperfect mechanisms for technology transfer:

Papers are usually written for peers and for posterity, rather than for anything approaching mass communication. The dissemination of knowledge in scientific disciplines is imperfectly understood, but it appears to require only a very small number of diligent readers to start the human networking process that eventually socializes the information in an important paper. Many other papers never get socialized at all and pass unnoticed into the archival purgatory [7, p. 119].

While Nochur and Allen found that the assigned gatekeeper was somewhat effective in importing new technology, he or she could not continue the transfer by moving it to other internal organizations (in essence acting like the umbrella gatekeeper). The third of these gatekeeper roles was most ineffective. Technology generally has to be wanted by the new organization and cannot be dictated by outsiders. However, the umbrella gatekeeper was really misclassified by Nochur and Allen. It is not a people mover strategy, since it is not a technology importation process, but instead represents an exportation of technology from one organization to another. We will call this new model *the rule model*. This method uses

an outside organization to impose a new technology on the development organization, which then infuses it into its own development process.

There are many examples within the government sector of this last technology transfer model. The mandating of the Ada language by the Department of Defense's Ada Joint Program Office for system development, the use of the Software Engineering Institute's Capability Maturity Model to evaluate developer's qualifications for a Department of Defense contract, the similar process of using international standard ISO 9000 in Europe, and the use of Federal Information Processing Standards (FIPS) by the National Institute of Standards and Technology (NIST) are all examples of technology transfer imposed by an outside agency.

Industry is not immune to the rule model either. Organizations have imposed tool standards on their subunits (e.g., imposing particular hardware and operating system products, CASE tools, or database systems), and organizations need to react to rules imposed by government contracting organizations, such as Request for Proposals that mandate a particular language, such as Ada. But in practice, however, successful examples of this imposed technology are rare. (It is not even clear if the above instances are successful examples of imposed technology transfer.) However, it is a model of technology transfer that has tremendous impact and we must not lose sight of it in our study.

3) *Advocates*: Fowler and Levine at the Software Engineering Institute have been investigating technology transition and have identified an extension to the gatekeeper model [6]. In their model, technology transition is a push-pull process:

Producer \Rightarrow Advocate \Rightarrow Receptor \Rightarrow Consumer.

The produce of the technology needs an advocate to export the technology outside of the development organization, while the consumer organization must have receptors agreeable to importing the technology. In many instances, however, both the advocates and receptors are part of the consumer organization, and in practice, this reduces to a model very much like Allen's gatekeeper.

4) *Successful Technology Transition*: It should also be realized that the PLC S-curve is the result of the introduction of a successful product, plotted after its success. There is no guarantee that a new product will follow this curve and most new products do indeed fail. This is the problem of most forecasts about the predicted growth in a new technology [14]. Developers of a new technology are almost always too optimistic about the eventual success of that technology. One of the goals of our study is to understand the relationship between those early users of a technology and the methods used to transform it into a mature technology.

B. Technology Maturation.

In 1985, Redwine and Riddle [11] published the first comprehensive study of software engineering technology transfer, which they called *maturation*. Their goal was to understand the nature of technology maturation—what was the length of time required for a new concept to move from being a

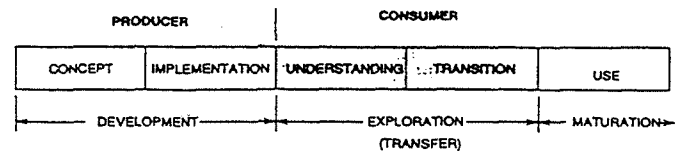


Fig. 2. Technology maturation life cycle.

laboratory curiosity to general acceptance by industry. They defined maturation of a technology as a 70% usage level across an industry.

Technology maturation involves five stages, two by the producer of the technology and three by consumers of that technology (Fig. 2).

- 1) The original *concept* for the technology appears as a published paper or initial prototype implementation.
- 2) The *implementation* of the technology involves the further development of the concept by the originator or successor organization until a stable useful version is created.
- 3) In the *understanding* stage, other organizations experiment, tailor, expand, modify, and try to use the technology.
- 4) In the later *transition* stage, use of the technology is further modified and expands across the industry.
- 5) The final *maturation* stage is reached when 70% of the industry uses the technology.

In their study, they looked at 17 software development technologies from the 1960's through the early 1980's (e.g., UNIX, spreadsheets, object-oriented design (OOD), etc.). Their results, most related to this current study, were the following:

- 1) They were unable to clearly define "maturation" for most technologies, but were able to make reasonable estimates as to the length of time needed for new technologies to become widely available.
- 2) Technologies required an average of 17 years to pass from an initial concept to a mature product.
- 3) Technologies, once developed, required an average of 7.5 years to become widely available (i.e., the Exploration stage of Fig. 2).

We view this current paper as the inverse of the Redwine-Riddle study. For each technology, how long did it take to infuse that technology into a given organization? That is, what was the Exploration stage within a given organization? The 7.5 years needed to penetrate an industry that was specified by the Redwine-Riddle study would be an upper bound, and we know that the lower bound is more than the gatekeeper simply declaring the new technology to be good.

III. TECHNOLOGY INFUSION

In order to understand technology transfer within NASA, about 15 software engineering experts at several NASA center were interviewed to determine which software engineering techniques were being used effectively in the agency. To keep the scope of this problem manageable, the following two restrictions were imposed:

TABLE I
TRANSFERRED TECHNOLOGIES AT NASA

	NASA Consumers	External Consumers
NASA Producers	Reuse(Kaptur), *AI(CLIPS) *GUI(TAE), CASE tools Measurement(SME, GQM)	Reuse(Kaptur), *AI(CLIPS) *GUI(TAE) Measurement(SME, GQM)
External Producers	Rate monotonic scheduling, CASE tools Cost models, *Formal Inspections *OOT *Ada, C, C++, *Cleanroom	Not relevant to this study

*—technologies discussed in more detail

- 1) The technology had to clearly be software engineering. For example, successfully transferred programs, such as the widely-used modeling system NASTRAN available through the NASA Software Repository (COSMIC), were not included.
- 2) The technology had to have a major impact on several groups within NASA. With more than 4000 software professionals affiliated with GSFC alone (including government and contractors), almost every software product has probably been used somewhere in the agency. While this was somewhat subjective, a list of transferred technologies was developed (Table I). Technologies developed outside of NASA and not used within NASA were outside of the scope of this study, hence the *Not relevant to this study* section of the table.

A. Examples of Technology Infusion

In this section we first discuss three technologies that were successfully infused into the state of the practice at GSFC (Ada, object-oriented technology (OOT), and Cleanroom) in greater detail. The details of transferring those technologies are summarized by Fig. 3. Each represents the understanding and transition stages as NASA plays the role of consumer organization trying to adopt these new technologies. These technologies were studied by the Software Engineering Laboratory (SEL) at GSFC. In addition, we also include several technologies transferred by groups other than the SEL.

The NASA/GSFC SEL has been a major source of technology infusion at Goddard Space Flight Center. The SEL was organized in 1976 to study flight dynamics software, and since that time it has had a significant impact on software development activities within the Flight Dynamics Branch. Most of these technologies (e.g., measurement, resource estimation, testing, process improvement) have been reported elsewhere [2].

As a brief overview of SEL operations, the SEL has collected and archived data on over 125 software development projects. The data are also used to build typical project profiles against which ongoing projects can be compared and evaluated. The SEL provides managers in this environment with tools for monitoring and assessing project status. Typically there are six to ten projects simultaneously in progress in the flight dynamics environment. Each project is considered an experiment within the SEL, and the goal is to extract detailed

information to understand the process better and to provide guidance to future projects.

Projects range in size from approximately 10K lines of source code to 300–500K at the high end. Projects involve from six to 15 programmers and typically take from 12 to 24 months to complete. All software was originally written in FORTRAN, but Ada was introduced in the mid-1980's, and there is now an increase in C and C++ programming.

1) *Use of Ada:* Ada is a language that was developed by the U.S. Department of Defense from 1976 to 1983 as a common language on which to build complex embedded applications. It is a general purpose programming language adaptable to any computing environment, although some claim that the language is too complex to use effectively. During the 1980's, as Ada compilers started to appear commercially, many organizations evaluated the language as a solution for their existing programming needs.

2) *Technology Transfer Model:* Use of Ada on flight dynamics projects was first considered in 1985. Because of Department of Defense interest in the language and because of NASA Johnson Space Center's decision to use Ada for Space Station software, the SEL desired to look at its applicability for other NASA applications. The initial stimulus for this activity, then could be a mixture of the communication model (i.e., papers were written about Ada), on-the-shelf model (i.e., Ada products were being sold), and to some extent, the rule model (i.e., since Johnson Space Center adopted Ada, there was some pressure to do the same elsewhere within NASA).

3) *Understanding Phase of Technology Transfer:* A training class and sample program was the first Ada activity. However, to truly evaluate the appropriateness of Ada within the SEL environment, a parallel development of an Ada (GRODY) and FORTRAN (GROSS) simulator was undertaken. GROSS, as the operational product, had higher priority and was developed on time. GRODY, as an experiment to learn Ada, had a much longer development cycle. In addition, since GRODY was known by all to be an experiment, the development team was not as careful in its design. However, the experiences of the GRODY team with the typical set of requirements NASA used for such products led to a greater interest in applying OOT instead as a model for future NASA requirements and design specifications. Although the development of this simulator continued until early 1988, by early 1987 it was decided that the initial project was sufficiently successful to continue the investigation of Ada on other flight dynamics problems. Elapsed time since start of Ada activity was 30 months.

Experiences at NASA Langley Research Center were similar to those of the SEL, but had a different conclusion. Understanding Ada began under the advocacy of one individual. A project was developed in both FORTRAN and Ada. Although the Ada project was deemed more successful than the FORTRAN version, the difference was not deemed great enough to enforce Ada on all projects.

4) *Transition Phase of Technology Transfer:* Because of the poor performance on the GRODY simulator and the problems with developing Ada requirements, the SEL undertook a second Ada pilot project (GOADA) as an

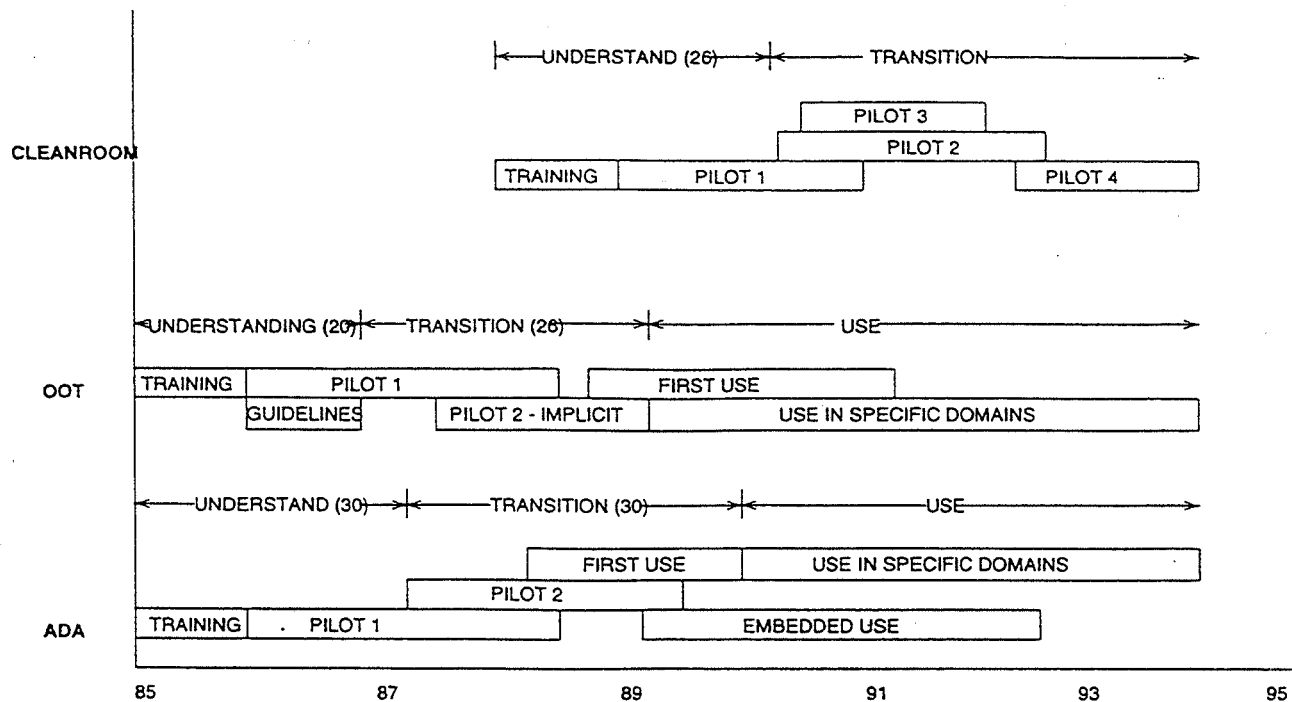


Fig. 3. SEL technology transfer experience.

experiment. However, there was sufficient confidence in Ada by this time to make GOADA an operational product, thus schedules and performance were more critical than with the previous GRODY experiment. In this case, the resulting product was comparable to performance of previous FORTRAN simulators. Between 1988 and 1990, four other simulators (one dynamic and three telemetry) were built successfully. In addition, one embedded application was developed beginning in 1989 and was not as successful due to the poor quality of the compilers for embedded applications that were available. By early 1990, Ada became the language of choice for simulators in the Flight Dynamics Division. Transition time was another 30 months.

5) *Comments on Technology Transfer:* Total transfer time for Ada was approximately 60 months. Ada is now the language of choice for simulator projects using Ada on DEC VAX computers. Although Ada code costs more, line by line, than FORTRAN code (about 20%), the higher levels of reuse result in lower overall delivery costs for such projects.

Ada was also proposed as the implementation language for large mission ground support systems, but this was never tested. The inhibitors in this case were outside of the features of the Ada language, itself. The operational systems at GSFC are IBM mainframe compatible, and no effective Ada compiler existed for this environment during the three times Ada was evaluated during the late 1980's. All of the successful simulator projects were implemented on DEC VAX computers, which did have an effective Ada system.

Presently, Ada is used on approximately 15% of the SEL's software. Eleven operational Ada projects have been completed to date. Another report gives a more complete analysis of the SEL's experiences with Ada [3].

One difficulty in evaluating the effects of Ada on software development within the SEL is due to side effects which may occur. The following study of OOT grew directly out of the early Ada experiences.

B. Object-Oriented Technology

In traditional software design of the 1960's and 1970's, a program consisted of a series of functions grouped into a set of subprograms. Software design consisted of developing these subprograms via functional decomposition of the overall program requirements into smaller functional units.

In the 1980's, OOD became an alternative to the earlier functional decomposition. A program now consists of a set of data objects and a set of operations that apply to these data objects. Software design now consists of developing these complex data objects (called data abstractions) and building a set of functions that manipulate the objects. Since design is more localized to the operations applying to a single data object, proponents of the method claim that the resulting product is more manageable, simpler, and more reliable.

1) *Technology Transfer Model:* Use of OOT in the SEL was investigated at the same time as Ada was considered, although was not a primary goal of the original decision to study Ada. In modifying the "standard" requirements of the FORTRAN-implemented GROSS simulator for the GRODY experiment (the simulator to be written in Ada), it became apparent that the standard GSFC requirements document was oriented toward a FORTRAN functional decomposition and the use of these requirements on an Ada project would be very inefficient. We view this as an example of the communications model of technology transfer. The existing requirements were

TABLE II

Statements	Ada Statements/h	FORTTRAN Statements/h
New statements	1.1	1.2
Reused Statements	5.0	5.5

deemed inadequate, and papers in the literature were collected on an alternative technology (i.e., object-orientation) which might be applicable in this domain.

2) *Understanding Phase of Technology Transfer:* Object orientation seems more natural an approach with the use of Ada packages and generic functions. Therefore the requirements for GRODY were rewritten to use a more object-oriented approach. Following this, an OOT guidebook for GSFC was developed General Object Oriented Software Development (GOOD) [13] for use on future projects.

The elapsed time for these activities lasted from early 1985 until August 1986, or a total of 20 months. Expenses for understanding this technology were high since this activity was wrapped up in the Ada evaluation which required parallel system development of GRODY with the FORTRAN equivalent GROSS.

3) *Transition Phase of Technology Transfer:* On a second project (UARSAGSS), OOD was used implicitly. This was a FORTRAN ground support system, and experiences gained from the earlier GRODY effort allowed the programmers to better understand the design and use OOT. By the end of this project, it was sufficiently clear that OOT was an effective technique in some domains. Transition time was on the order of 26 months.

4) *Comments on Technology Transfer:* Total transfer time in this case was only 46 months. Although almost four years, this was relatively short since it did not require major changes in system development. The same set of tools could be used: OOT was mostly a change in approach toward system building that could be used with any underlying implementation language. Although initially considered as an Ada technique, the same methods would map easily to a FORTRAN development model. Since it fit within the usual development paradigm, tailoring the method and inserting it into the usual NASA development process was relatively easy.

It should also be mentioned, that although OOT was originally studied within the Ada domain, it has had a profound effect on productivity on FORTRAN projects as well. This provides an additional reason why adoption of Ada as the development language has not provided significantly better productivity within the SEL. Although overall productivity using Ada has greatly improved over FORTRAN productivity of the mid-1980's, FORTRAN productivity has also improved dramatically.

For example, productivity measurements in statements produced per hour of effort on four recent Ada and four recent FORTRAN projects show that FORTRAN is still easier to write and is easier to reuse than Ada code [15]. (See Table II.)

While not statistically significant, the data does indicate that both languages seem comparable.

TABLE III

Language	1988-1990 % Reuse	1990-1994 % Reuse
Ada	4-17 (three projects)	64-88 (four projects)
FORTTRAN	4-12 (seven projects)	75-90 (four projects)

Reuse has proven to provide the largest boost in productivity with both Ada and FORTRAN due to the effects of OOT. (See Table III.)

Although not every project achieves such high levels of reuse, the trend is certainly upwards. This shows the serendipity nature of technology transfer. You generally cannot dictate exactly what you want to change, and change may occur from unexpected sources.

C. Cleanroom

Cleanroom is a software development method that relies on *a priori* verification of a software product rather than the usual *a posteriori* testing of software for validation. All software designs are verified via both formal and informal proofs of correctness rather than relying on later testing to find errors. The claim is made that errors are easier to find within a design document before that design has been codified into a large source program. The method was developed by H. Mills while at IBM during the late 1970's.

The immediate effect of this method is that a project spends more time and effort in the design phase as designs are verified than with more traditional development methods. The payoff is that less testing will be needed later, with a decrease in total project costs and an increase in product reliability.

1) *Technology Transfer Model:* Cleanroom was studied in the SEL via the people-mover model, at the instigation of V. Basili of the University of Maryland, who is also one of the SEL Directors. Previously, Basili had studied Cleanroom within a student environment at the University of Maryland.

2) *Understanding Phase of Technology Transfer:* To understand Cleanroom, a series of training courses was given in 1988 by H. Mills, the original developer of the method. A pilot project was undertaken and proved to be very successful. All participants were converts to the method, even though several had reservations about it before they began. The time to understand the method (training until the start of the second Cleanroom project) was 26 months.

3) *Transition Phase of Technology Transfer:* Two follow-on Cleanroom projects were undertaken. A smaller in-house development was very successful, but a larger contracted project was not so successful. It was not as apparent that problems on the larger project were due to scaling up of Cleanroom to larger tasks or to a lack of training and motivation of the development team on this project. For this reason, a fourth larger Cleanroom project was undertaken. The fourth pilot was completed in early 1995 (after this technology transfer study officially ended) and the resulting system was considered to be extremely reliable.

4) *Comments on Technology Transfer:* Cleanroom technology appears to be an effective technology on smaller projects, but may lose some of its effectiveness on NASA

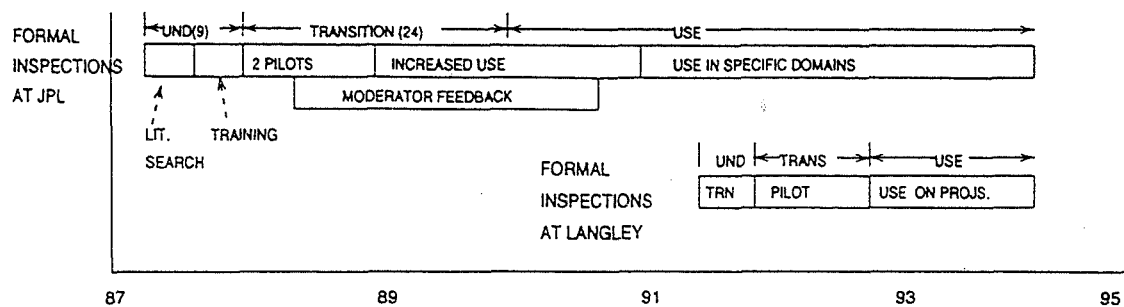


Fig. 4. Formal inspections technology transfer experience.

projects involving more than 160K lines of code. Training and motivation of the staff were considered crucial for its success. Understanding time was 26 months and transition time was about 46 months. With the completion of the fourth pilot project, the Flight Dynamics Branch is now evaluating Cleanroom's role in future developments.

D. Formal Inspections

In addition to the techniques evaluated by the SEL, technology infusion was studied elsewhere within NASA. One such technique whose use is growing within NASA is the use of formal inspections (Fig. 4).

A formal inspection is a verification method that has aspects similar to the Cleanroom method mentioned previously. In a formal inspection, one software artifact (e.g., a module design, source code for a module, test data) is identified for inspection. The author of that artifact must present the design of that artifact to a meeting of fellow workers, who were previously given the artifact to study. The process of discussing the artifact for a given period of time (generally not more than 2 h) has proven to be very effective in finding errors in the object of study.

1) *Infusion of Formal Inspections:* We first studied the infusion of formal inspections within the Jet Propulsion Laboratory (JPL).

2) *Technology Transfer Model:* Initial work at the JPL on formal inspections began in February 1987 in response to a need for higher quality software. J. Kelly of JPL was the initial gatekeeper who proposed studying this technology.

3) *Understanding Phase of Technology Transfer:* In mid-1987, after studying the literature, a decision was made to tailor inspections for JPL use. A course was developed for use at JPL and the initial advocates for inspections sought other JPL managers (the receptors) who would use and benefit from the technology.

4) *Transition Phase of Technology Transfer:* In 1988, approximately two or three pilot projects were started using inspections. Bimonthly moderator meetings were held to spread the advocacy from the initial developers to other managers at JPL so that there would be others who "own" the process should the developers leave the organization. From 1989 to 1991 additional projects used the technology to help institutionalize the process. By late 1989 it was rapidly becoming a standard technology at JPL.

5) *Comments on Technology Transfer:* The elapsed time for developing the method was about 33 months and involved about three staff years of effort. Meetings and telephone contact with M. Fagan of IBM, developer of the technique, helped the JPL staff understand the process. It has been successfully transferred to JPL and between its initial use in February 1987 through 1990, over 200 inspections were carried out. The use of moderator meetings greatly aided the "people mover" model as other managers became advocates of the technology to help infuse it at JPL.

6) *Transfer of Formal Inspections:* Based upon experience at JPL, NASA tried to move the technology to other NASA centers.

7) *Technology Transfer Model:* The transfer of formal inspections demonstrates the difficulty of the assigned gatekeeper model described earlier. The need for this gatekeeper was quite apparent early in this process. For example, although meetings were held with management at Kennedy Space Center, no receptor for the technology was found and the infusion process never took hold (i.e., failure of the assigned gatekeeper role).

To avoid this problem at other centers, the transfer process first tried to identify the appropriate receptor who would promote the infusion process. At Langley Research Center, a course was offered and an advocate was identified who helped with the infusion. In May 1991 the initial Formal Inspection course was offered, and by Fall 1991 a pilot project was started. In August 1992, inspections were declared a standard part of all developments. This process took only 16 months, because of the previous experiences at JPL. About 12 staff months of effort were required, but most of this effort was in "unpaid overtime." No NASA support was available for developing the technology. Once installed at Langley, it has been transferred to several contractors working at Langley.

8) *Comments on Technology Transfer:* Formal inspections were successfully transferred at JPL and Langley. Total time for transferring at both centers were 33 months and 16 months. These were relatively short since formal inspections cover only a relatively narrow and precise process in software development and can be inserted relatively easily into almost any mature development process. On the other hand, the process was not successful at other centers where no advocate was found.

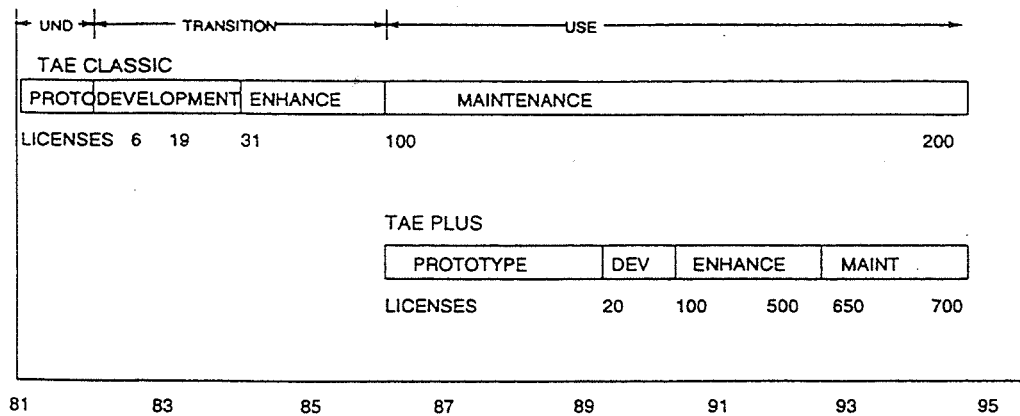


Fig. 5. TAE development.

E. Transportable Application Environment (TAE)

The transportable application environment (TAE) probably represents NASA's major success in exporting software products outside of NASA. TAE is a graphical user interface useful as a front-end for other software products. It was developed between 1981 and 1990 at GSFC.

Initially TAE was a simple character-oriented interface between the user at a terminal and an application. With the development of the X Window System by the MIT X Consortium in the 1980's, TAE was rewritten to use the X Windows graphical interface.

1) *TAE Development as a Producer:* Originally designed in 1981 (See Fig. 5) to support ASCII terminals, this product was renamed TAE Classic when an enhanced X-Windows and Motif compatible version (TAE Plus) was developed in 1986. It was originally distributed through the NASA software library COSMIC. Over 900 licenses have been obtained for the product. TAE represents one of the few commercial successes in software technology transfer for NASA. For the past two years non-NASA users have obtained TAE commercially via Century Computing, Inc. It is one of the few software engineering technologies that has been transferred via the on-the-shelf model of technology transfer, although the communication model was the primary vehicle for "getting the word out."

2) *Comments on Technology Transfer:* The TAE developers support its users via: 1) A help desk in lieu of training and consulting, 2) over 1500 pages of reference documentation, 3) newsletters giving advice on how to use the system, and 4) TAE user conferences held approximately every 18 months.

Exporting TAE encountered problems quite similar to those encountered in other technology transfer environments:

- 1) The cultural bias against anything new made other organizations reluctant to try this product. The "Not Invented-Here" syndrome is strong in software development organizations. Discovering potential gatekeepers from the outside is difficult to accomplish.
- 2) The mechanisms to make potential users aware of the benefits of TAE were inadequate.
- 3) Mandating use of an immature technology (the Rule model) may be counterproductive if it causes an effective

technique to be discarded before it is ready to be used. TAE Classic was not quite ready in the mid-1980's and its mandated use on some GSFC projects led to less than optimum performance and a lack of advocacy among some development groups. It took the development of TAE Plus to get the product to its full potential. Therefore, TAE use within GSFC generally lags behind use of TAE at other NASA centers.

- 4) There was a reluctance to accept a government-developed product as legitimate and of quality design. However, two-thirds of all TAE licenses are at non-NASA sites.

3) *TAE Development as a Consumer:* TAE use spread through NASA. TAE usage within NASA was similar to infusion of any other technology. Reports about TAE were read, and since the group was already committed to an X-Windows and Motif interface, use of TAE Plus seemed appropriate. Total understanding and transition time was approximately 42 months before it became operational.

F. CLIPS Expert System

- The C Language Integrated Production System (CLIPS) was developed at NASA/JSC (Johnson Space Center) as an expert system to solve problems that JSC was having with previous products [8]. Using a rule-based syntax similar to an automated reasoning tool (ART), CLIPS was implemented in C to be efficient and portable.

CLIPS development began around 1985 as a prototype proof-of-concept batch implementation that would use some of the syntax from the earlier ART product (Fig. 6). The initial goal was simply a training exercise to make JSC an "intelligent customer" for similar products. However, after completing the prototype, in May 1985 it was decided to build a product for internal use for the HP workstation. This product was completed around May 1986. Only after completion did the developers consider the possibility that others may want to use it, and a quality assurance (QA) phase was instituted to eliminate remaining errors and make the system portable and useful by others. The software was submitted to COSMIC in August 1986.

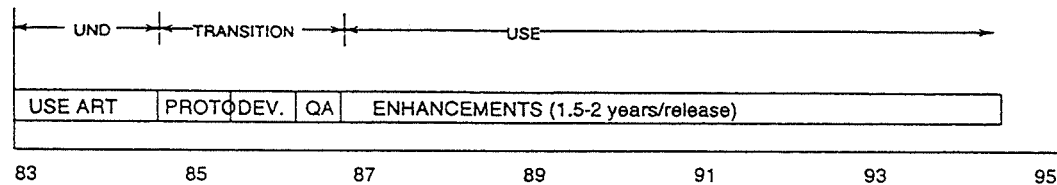


Fig. 6. CLIPS development.

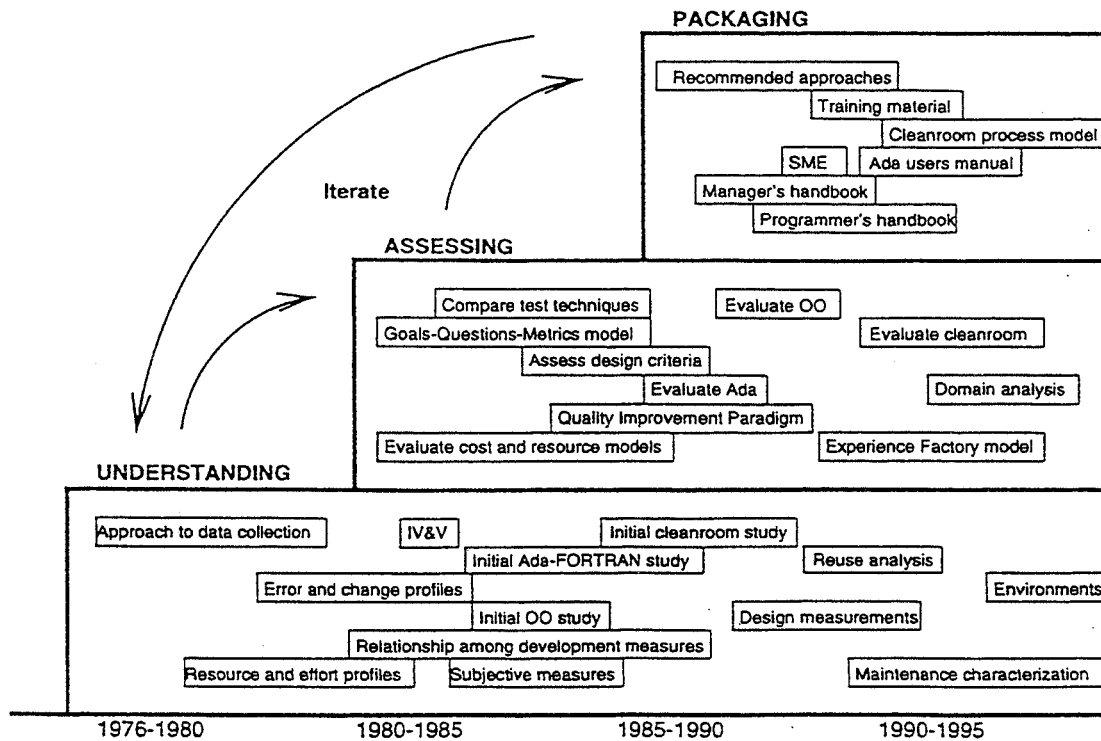


Fig. 7. Summary of SEL studies.

CLIPS had a relatively short 15-month development cycle. Since its rules were based upon the previous ART expert system, there was little need for design trade-off studies. The basic rule-based algorithms were coded in C for efficiency and portability. Since its initial release, versions of CLIPS have been ported to DEC VAX's, mainframes, UNIX workstations, Macintoshes, and PC's; its portability enhanced by being coded in C.

1) *Comments on Technology Transfer:* CLIPS was transferred outside of NASA/JSC by the following mechanisms:

- *NASA Technology Transfer facilities:* CLIPS was submitted to COSMIC for general distribution. Since 1986, CLIPS has over 5000 users in government, academia, and industry. COSMIC was viewed as just a vehicle for distributing the source program, but not in how to spread the word about the product. CLIPS represents a second example, like TAE, of an "on-the-shelf" product being exported outside of NASA.
- *Conferences:* Papers on CLIPS appeared at many professional conferences. Papers started to appear that referenced the use of CLIPS to solve various application problems, thus increasing interest in the product.

- *Technology transfer model:* The people mover model via "word of mouth" was the most effective means for distributing information about CLIPS. The JSC development group distributed copies to other NASA centers, where NASA project managers became advocates who wanted their contractors to use CLIPS for relevant applications. Thereafter, nonspace-related divisions of these contractors learned about CLIPS and started to use it on non-NASA applications. Discussions over the Internet spread the word about the product. Being written in C, its efficiency led to rapid growth in its use.

Both CLIPS developers and some users claimed that one of the reasons for its spread was that it was the only expert system shell generally available in source program format. This enabled users to tailor the product for their own local environment and made understanding aspects of the system easier.

G. Software Engineering Processes

Software processes are generally not embodied in a product. Transfer of processes turns out to be very difficult. For example, the most successfully transferred products found in

this study of NASA were TAE and CLIPS, both executable products. Transfer of processes (e.g., Cleanroom, OOT, formal methods) was found to be much more limited in this study.

1) *SEL Measurement Model*: To address this problem, the SEL has developed the quality improvement paradigm (QIP) for the diffusion of new innovation within an organization. Fig. 7 briefly summarizes this process as an evolution of SEL activities since the inception of the SEL in 1976. The process improvement model involves understanding a technology, assessing its applicability within a new environment, and packaging it for routine use.

The *understanding* step baselines process and product characteristics (e.g., cost, reliability, software size, reuse levels, error classes). The *assessment* step incorporates potential improvements into development projects that are to be evaluated. Quantified SEL experiences (e.g., most significant causes of errors) and clearly predefined goals for the software (e.g., decrease error rates) drive the selection of candidate process changes. After the changes are selected, training is provided and experiment plans are produced. Then the processes are applied to one or more production projects from which detailed measurements are taken. The new process is assessed by comparing these measures with the continually evolving baseline. As a result of the analysis, processes are adopted, discarded, or tailored for ensuing efforts depending on the observed impacts. The *packaging* step infuses identified improvements into the standard SEL process. This includes updating and tailoring standards, handbooks, training materials, and development support tools.

Process improvement applies to both the individual project or experiment level (observing two or three projects) as well as to the overall organization level (observing trends of numerous projects over many years). In the early years, the SEL emphasized building a clear understanding of the process and products within the environment. This led to the development of models, relations, and general characteristics of the SEL environment. Most of the experiments (process changes) consisted of the study of specific, focused techniques (e.g., program design notations, structure charts, reading techniques), but the major enhancement was the infusion of measurement, process improvement concepts, and the realization of the significance of the process as part of the software culture.

2) *Measurement and Experimentation Elsewhere*: One of the original goals of this study was to also collect data on the costs of technology infusion. However, outside of the SEL, very little data was collected on which to make any conclusions. While total project costs are usually collected by development organizations, there is usually no real breakdown into the various activities required for exploring a new technology. Any results here would be mostly unreliable guesses.

IV. CONCLUSIONS

In studying technology transfer we believe that we have identified the five models of technology transfer at work within NASA, the four models (people-mover, communications, on-the-shelf, and vendor) of Berniker as well as our fifth rule model.

Infusion of technology generally took from two to four years to accomplish. An initial study, training course, or prototype development took from six months to a year. Once the technology was deemed appropriate, two to four pilot studies were undertaken as the method was tailored to the local environment. After several of these studies, the method (either implicitly or explicitly) became state-of-the-practice within that organization.

One limitation to this study is that NASA, as a government agency, is not driven by the same set of market forces as in a profit-making organization. While pressure to downsize, lower costs, increase reliability, and shorten the development cycle are as true for NASA as they are for most other organizations, the demands to do so are driven more by management (and Congressional) demands and less by loss of market share and lack of profit.

From this study, we can make several observations about technology transfer mechanisms. While these conclusions apply solely to NASA, we believe that the results are fairly general and should apply to other comparable technological organizations.

A. Differences Between Software Engineering and Other Technologies

This study of NASA exhibited several attributes of software engineering which differ from other engineering disciplines. As such, software engineering technology infusion follows a process that differs somewhat from other technology transfer models:

- 1) *Infusion mechanisms do not address software engineering technologies well*. Software has a crucial difference that separates programming from other engineering disciplines. Just as software differs from other products in that it really does not "age," "decay," or "cease to function" (and hence engineering concepts like mean-time-to-failure are not truly relevant with software), the development of software is much more process-driven and less product-oriented. Software engineering is currently very dependent upon programmer expertise and less upon implemented technology than with many other forms of engineering, as we describe below.

There are few integrated systems that effectively aid the software engineer to build complex systems. Most software engineering technology are processes, a set of rules to be followed in the development of systems. How to package and transfer processes as a corporate asset must be handled better. For example, within the GSFC Software Engineering Laboratory, the following processes have been studied over the past few years:

- a) OOT;
- b) goals/questions/metrics paradigm of software development;
- c) the experience factory model of development;
- d) Cleanroom software development.

(Only a) and d) were described in this paper [2])

None of these processes is embodied in a product. One cannot buy a "Cleanroom" program. Instead one buys

some books, a training course, and some guidance on using the technique. Although NASA does not explicitly address the packaging of such processes as assets to be transferred, NASA is not unique in this regard. Much of industry does not understand the unique role that processes play in software development compared to most engineering processes. It is imperative for industry to understand this distinction and to address the transfer of processes as well as products.

This observation makes software engineering appear more like Allen's view of science rather than technology [1]. In science, the desired output is "verbally encoded information" in the form of published papers, whereas in technology, the desired result is "physically encoded information" in the form of hardware products. "Verbally encoded information" in the form of product documentation or published papers on the technology is not viewed as important as the product itself. Current technology transfer organizations are attuned to the technology model of technology transfer and have not adapted (or needed to adapt) to the *science* model of software engineering technology.

- 2) *Quantitative data is crucial for understanding software development processes.* Like other engineering disciplines, without quantitative results, it is impossible to fully understand what is happening and what are the effects of instituted changes. However, outside of the SEL at NASA/GSFC, few organizations (both inside NASA and outside) collect effective data on their development practices. In this study, while we were able to track the time to accomplish several instances of technology infusion, any details about the costs of such technology transfer and on their bottom-line effects on project costs, reliability, or schedules were mostly educated guesses. Measurement and experimentation are not part of the software development culture.
- 3) *Technology infusion is not free.* Organizations already understand that without the appropriate advocate already in place in some infusing organization, the ability to export a new technology to that organization depends upon a significant marketing effort to make the new organization aware of the benefits of the new technology. For example, NASA already spends considerable funds running a Technology Utilization Office, COSMIC, and other components of its technology transfer program. On the other hand, technology infusion is rarely supported.

Unlike other engineering disciplines, rarely are Internal Research And Development (IRAD) funds available for developing new software technology. New technology is often procured out of existing project funds and not capitalized over the life of the product. This may be related to the previous comment that software engineering behaves more like a science, and IRAD funds typically are used for technology developments.

Organizations generally keep track of total project costs, but separating them into individual tasks (pilot project development, tool use training, new method experimentation, etc.) is generally not done. The cost of

innovation as a part of an operational project becomes a severe inhibitor to using new innovations. Costs of such innovation have to be borne by project development budgets, greatly increasing the risk of a cost overrun. Conservative fiscal planning makes innovation an even higher risk than is necessary.

B. Similarities Between Software Engineering and Other Technologies

To a great extent, we reconfirmed within the software engineering domain many of the issues concerning technology transfer found by others:

- 1) *Most software professionals are resistant to change.* One manager at NASA referred to the "cultural block" to new technology by those who were used to mainframe computers, while another manager was more pragmatic in stating that his staff needed to see the technology demonstrated in a meaningful way in their own environment before they would consider accepting the technology.
- "Activities in this [technology transition] life cycle must attend to the culture of the organization in which the technology is being implemented" [6] is an aspect of technology infusion that is often ignored. The motto of "one size fits all" should not apply in this domain.
- 2) *Technology transfer is more than simply understanding the new technology.* Technology transfer takes time. Understanding the technology has been shown to take upwards of 2.5 years, and it usually involves multiple instances of training and pilot projects. The transition time when the organization is exploring, tailoring and modifying the technology for its own use often takes more than the understanding time, with a total transfer time on the order of five years not being unusual.
- 3) *Technology infusion is part of the total environment of the consumer organization.* Technology infusion does not occur in a vacuum. The SEL experience with Ada is such an example. Ada proved to be successful with flight simulators. However, the operational system for flight dynamic software was the IBM mainframe, and no effective Ada compiler was available during 1985-1990 when Ada was under evaluation. The "window of opportunity" on using Ada has passed and FORTRAN remained the language of choice for such applications. Had an effective mainframe Ada compiler been available, then the result of evaluating Ada for large systems might have been different.¹
- 4) *The government can have an impact on technology infusion.* As others have shown, the imposition of rules mandating certain technologies (i.e., the rule model) is generally not very effective. However, the experiences with inspections and CLIPS demonstrates that the government can employ an effective people mover strategy

¹ For the past several years the use of C and C++ has been growing within this community, and it now looks like FORTRAN may be replaced by C++ as the language of choice for flight dynamics applications.

to infuse technology. In both of these cases, advocates were recruited from NASA centers different from the development group. Those advocates started to use these technologies with contractors working for them. The transfer process occurred as other development groups within the contractor organization noticed the technologies that their colleagues were using and then voluntarily decided that they were effective for solving certain problems. Project by project, these technologies gradually spread among the contractors without the need for mandates.

- 5) *People contact is the main transfer agent of change.* As many others have observed, technology transfer occurs best when the developers of a technology are involved in the technology transfer process. In our study, that happened in order for Cleanroom to be effectively used at GSFC, for inspections to be brought first to JPL and then to Langley, and for CLIPS to develop an initial set of users. Finding the appropriate advocate to act as gatekeeper for the technology is a crucial component of any technology transfer mechanism.
- 6) *Timing is a critical decision.* This can be either a positive or a negative influence. When to enforce a decision is important for its adoption. The TAE experience at GSFC shows that early mandating of an immature technology may have the paradoxical consequence of delaying an effective technology even more than by not mandating it at all.

On the other hand, the early studies of Ada led to the observation that OOT might have an impact on the organization. The result of this observation was an improvement in the use of Ada as well as vastly improved FORTRAN code being produced.

Technology infusion today is generally ignored and left up to the individual engineer to discover what is needed and available. With today's shrinking budgets and the need to work "better, faster, cheaper," management needs to address this issue and help in the search for new effective technology to use.

ACKNOWLEDGMENT

The author wishes to thank F. McGarry of Computer Sciences Corporation (formerly of NASA/GSFC) and K. Jeletic of NASA/GSFC for helping to collect the data described in this paper.

REFERENCES

- [1] T. J. Allen, *Managing the Flow of Technology*. Cambridge, MA: MIT Press, 1977.
- [2] V. Basili et al., "SEL's software process-improvement program," *IEEE Software*, pp. 83-87, Nov. 1995.
- [3] J. Bailey, S. Waligora, and M. Stark, "Impact of Ada in the flight dynamics division: Excitement and frustration," in *18th Software Eng. Workshop*, SEL-93-003, NASA/GSFC, Dec. 1993, pp. 422-449.
- [4] V. R. Basili, "The experience factory: Can it make you a 5?" in *17th NASA/GSFC Software Engineering Workshop*, Greenbelt, MD, Dec. 1992, pp. 55-64.
- [5] E. Berniker, "Models of technology transfer (A dialectical case study)," in *IEEE Conference: The New International Language*, Portland, OR, July 1991, pp. 499-502.
- [6] P. Fowler and L. Levine, "A conceptual framework for software technology transition," Tech. Rep. SEL-93-TR-31, Software Eng. Institute, Carnegie Mellon Univ., Dec. 1993.
- [7] R. W. Lucky, *Silicon Dreams: Information, Man and Machine*. New York: St. Martin's, 1989.
- [8] W. Mettrey, "A comparative evaluation of expert system tools," *IEEE Software* vol. 24, no. 2, pp. 19-31, Feb. 1991.
- [9] K. S. Nochur and T. J. Allen, "Do nominated boundary spanners become effective technological gatekeepers?," *IEEE Trans. Eng. Manage.*, vol. 39, pp. 265-269, Aug. 1992.
- [10] M. C. Paulk, B. Curtis, M. B. Chrissis, and C. V. Weber, "Capability maturity model for software," Tech. Rep. SEL-93-TR-24, Vers. 1.1, Software Engineering Inst., Pittsburgh, PA, 1993.
- [11] S. Redwine and W. Riddle, "Software technology maturation," in *8th IEEE/ACM Int. Conf. Software Eng.*, London, U.K., Aug. 1985, pp. 189-200.
- [12] E. Rogers, *Diffusion of Innovation*. New York: Free Press, 1983.
- [13] E. Seidewitz and M. Stark, "General object oriented software development," Tech. Rep. SEL-86-002, NASA/GSFC, Aug. 1986.
- [14] S. Schnaars, *Megamistakes*. New York: Free Press, 1989.
- [15] S. Waligora, J. Bailey, and M. Stark, "The impact of Ada and object-oriented design in the Flight Dynamics Division at Goddard Space Flight Center," Tech. Rep. SEL-95-001, NASA Goddard Space Flight Center, Mar. 1995.
- [16] M. V. Zelkowitz, "Assessing software engineering technology transfer within NASA," Tech. Rep. NASA-RPT-003-95, NASA Software Eng. Program, Jan. 1995.



Marvin V. Zelkowitz (M'72-SM'78) received the B.S. degree in mathematics from Rensselaer Polytechnic Institute, Troy, NY, and the M.S. and Ph.D. degrees in computer science from Cornell University, Ithaca, NY, in 1967, 1969, and 1971, respectively.

He is a Professor of Computer Science at the University of Maryland, College Park, where he also holds a joint appointment with the university's Institute for Advanced Computer Studies. Since 1976 he has also held a faculty appointment with the Computer Systems Laboratory of the National Institute of Standards and Technology (NIST). His research interests include environment design and the effects of software engineering technologies in practice.

Dr. Zelkowitz is a member of ACM. He was Chairman of the IEEE Computer Society's Washington Chapter, ACM SIGSOFT, and the Computer Society's Technical Committee on Software Engineering.

Communication and Organization in Software Development: An Empirical Study

Carolyn B. Seaman
Victor R. Basili

*University of Maryland
Institute for Advanced Computer Studies
Computer Science Department
College Park, MD, USA*

52-61
415768

Abstract

The empirical study described in this paper addresses the issue of communication among members of a software development organization. The independent variables are various attributes of organizational structure. The dependent variable is the effort spent on sharing information which is required by the software development process in use. The research questions upon which the study is based ask whether or not these attributes of organizational structure have an effect on the amount of communication effort expended. In addition, there are a number of blocking variables which have been identified. These are used to account for factors other than organizational structure which may have an effect on communication effort. The study uses both quantitative and qualitative methods for data collection and analysis. These methods include participant observation, structured interviews, and graphical data presentation. The results of this study indicate that several attributes of organizational structure do affect communication effort, but not in a simple, straightforward way. In particular, the distances between communicators in the reporting structure of the organization, as well as in the physical layout of offices, affects how quickly they can share needed information, especially during meetings. These results provide a better understanding of how organizational structure helps or hinders communication in software development.

1. Introduction

Software development managers strive to control all of the factors that might impact the success of their projects. However, the state of the art is such that not all of these factors have been identified, much less understood well enough to be controlled, predicted, or manipulated. One factor that has been identified [Curtis88] but is still not well understood is information flow. It is clear that information flow impacts productivity (because developers spend time communicating) as well as quality (because developers need information from each other in order to carry out their tasks well). The study described in this paper addresses the productivity aspects of communication by empirically studying the organizational and process characteristics which influence the amount of effort software developers spend in communication activities. This is a first step towards providing management support for control of communication effort.

This research also arises out of an interest in the organizational structure of software enterprises and how it affects the way software is developed. Development processes affect, and are affected by, the organizational structure in which they are executed.

This work was supported in part IBM's Centre for Advanced Studies, and by NASA grant NSG-5123.

Communication in software development is one area in which organizational and process issues are intertwined. A process requires that certain types of information be shared between developers and other process participants, thus making information processing demands on the development organization. The organizational structure, then, can either facilitate or hinder the efficient flow of that information.

The empirical study described here aims to identify the organizational characteristics which affect process communication effort, and to determine the degree of effect. The dependent variable in this study is communication effort, defined as the total effort expended to share some type of information. The dependent variables are organizational distance, physical distance, and familiarity. All three of these are measures of the organizational structure, defined as the network of relationships between members of the software development organization. The types of relationships upon which these measures are based are, respectively, official relationships, physical proximity, and past and present working relationships.

The study combines quantitative and qualitative research methods. Qualitative methods are designed to make sense of data represented as words and pictures, not numbers [Gilgun92]. Qualitative methods are especially useful when no well-grounded theories or hypotheses have previously been put forth in an area of study. Quantitative methods are generally targeted towards numerical results, and are often used to confirm or test previously formulated hypotheses. They can be used in exploratory studies, but only where well-defined quantitative variables are being studied. We combine these paradigms in order to flexibly explore an area with little previous work, as well as to provide quantified insight that can help support the management of software development projects.

The purpose of this report is twofold. First, we wish to describe the methods we have used to carry out this study, so that other researchers can consider their appropriateness for investigating this area. Second, we wish to present a set of useful results, in order for practitioners and others to gain more understanding of communication and organizational issues in software development projects. In the subsections which follow, the specific problem addressed by this study is presented, as well as the research questions and some definitions of terms. In section 2, the related work in the literature is outlined. Our research methods are described in detail in section 3, and section 4 presents our results. In section 5, some of the limitations of this study are presented and packaged as experience to be used in future efforts to address this issue. Finally, section 6 discusses and summarizes the results of the study.

1.1. Problem Statement

Software development organizations do not currently know how to ensure an efficient flow of information among developers. They do not know how to assess, with any certainty, the information flow requirements of the development processes they choose. In addition, they do not have a deep understanding of how their organizational context affects the level of effort needed to meet the process's communication requirements.

The lack of understanding of communication issues has several consequences. First of all, managers have no way to account for communication costs in their project planning, or to balance those costs with the benefits of communication. Additionally, they do not know how to identify or solve communication problems when they arise. Finally, we cannot begin to learn from experience about communication issues until we identify the important variables that affect communication efficiency.

1.2. Research Questions

The study of organizational issues and communication in software development is not advanced to the point where it is possible to formulate well-grounded hypotheses. Therefore, this work is based on the following set of research questions:

- How does the distance between people in the management hierarchy of a software development organization affect the amount of effort they expend to share information?
- How does a group of software developers' familiarity with each other's work affect the amount of effort they expend to share information?
- How does the physical distance between people in a software development organization affect the amount of effort they expend to share information?

These questions are all operationally specialized versions of the more general question:

- How does the organizational structure in which software developers work affect the amount of effort it takes them to share needed information?

These research questions lead directly to a set of dependent and independent variables for the study proposed in this document. The dependent variable is *Communication Effort*, defined as the amount of effort expended to complete an interaction. Secondly, there is a set of independent variables which represent organizational structure. Three different measures have been chosen which capture the different properties mentioned in the first three research questions above. The first, *Organizational Distance*, measures the distance between people in the official management structure of the development organization. The second is *Familiarity*, which reflects how familiar different developers are with each others' past and present work. Finally, the independent variable *Physical Distance* is a measure of physical proximity. The proposed study will explore the relationship between each of these three independent variables, and the dependent variable.

The study design also includes a large set of intervening, or blocking, variables. These factors are believed to have an effect on communication effort, but are not the primary concern of this study.

1.3. Definitions

In this section, some important concepts are defined in the context of this study.

organizational structure - the network of all relevant relationships between members of an organization. These relationships may affect the way people perform the process at hand, but they are not defined by the process being performed.

process - a pre-defined set of steps carried out in order to produce a software product.

process communication - the communication, between members of a development project, required explicitly by a software development process.

communication effort - the amount of effort, in person-minutes, expended to complete an interaction, including the effort spent requesting the information to be shared, preparing the information, transmitting or transferring the information from one party to another, and digesting or understanding the information. This definition includes effort spent on activities not normally considered "communication" activities (e.g. preparing and reading written information).

interaction - an instance of communication, in which two or more people are explicitly required (by the process they are executing) to share some piece of

information. For example, the handoff of a coded component from a developer to a tester is an interaction. One developer asking for advice from an expert, no matter how crucial that advice may be, is not an interaction according to our definition. An interaction begins when some party requests information (or when a party begins preparation of unrequested information) and ends after the information has been received and understood sufficiently for it to be used (e.g. read).

qualitative data - data represented as words and pictures, not numbers [Gilgun92].

Qualitative analysis consists of methods designed to make sense of qualitative data.

quantitative data - data represented as numbers or discrete categories which can be directly mapped onto a numeric scale. **Quantitative analysis** consists of methods designed to summarize quantitative data.

participant observation - research that involves social interaction between the researcher and informants in the milieu of the latter, during which data are systematically and unobtrusively collected [Taylor84].

structured interviewing - a focused conversation whose purpose is to elicit responses from the interviewee to questions put by the interviewer [Lincoln85].

coding - a systematic way of developing and refining interpretations of a set of data [Taylor84]. In this work, coding refers specifically to the process of extracting specific pieces of information from qualitative data in order to provide values for quantitative research variables.

triangulation - the validation of a data item with the use of a second data source, a second data collection mechanism, or a second researcher [Lincoln85].

member checking - the practice of presenting analysis results to members of the studied organization in order to verify the researcher's conclusions against the subjects' reality [Lincoln85].

2. Related Work

The work proposed in this document is supported by the literature in three basic ways. First of all, the research questions in section 1.2 have been raised in various forms in the literature. The relationship between communication and organizational structure (in organizations in general) is a strong theme running through the organization theory literature, from classic organization theory [Galbraith77, March58], to organizational growth [Stinchcombe90], to the study of technological organizations [Allen85], to business process reengineering [Davenport90, Hammer90]. This relationship has not been explored in detail in software development organizations. However, several studies have provided evidence of the relevance of both organizational structure (along with other "non-technical" factors) and communication in software development. In particular, at least one study [Curtis88, Krasner87] points to the three aspects of organizational structure which we address in our research questions.

Second, our chosen dependent and independent variables have all appeared in some form in the literature. Our dependent variable, Communication Effort, has been defined to include both technical and managerial communication, to reflect only that communication required by the development process, but to include *all* such process communication. These decisions are based on results presented in the organization theory [Ebadi84, MaloneSmith88] and empirical software engineering literature [Ballman94, Bradac94, Perry94]. The three independent variables also appear in these two areas of literature. Organization theory points to the benefits of organizational and physical proximity of communicators [Allen85, Mintzberg79], while empirical software engineering has shown the drawbacks of organizational and physical distance [Curtis88]. The idea of "familiarity"

is referred to in a more general way in the literature. Both areas refer to the importance of various types of relationships between communicators [Allen85, Curtis88]. In particular, a software development study [Krasner87] has discovered the importance of "shared internal representations", which have led to our particular definition of Familiarity.

Third, the literature in many areas has helped shape the design of the proposed study by providing methods and experience. The choice and definition of the unit of analysis, the interaction (section 3.2), has been influenced by the organization theory [Ebadi84, Liker86] and empirical software engineering literature [Perry94]. The scope of the study, in terms of the types of communication studied, has also been influenced by this literature. Data collection and analysis methods have come directly from the literature in empirical methods [Lincoln85, Taylor84].

Despite the considerable support in the literature, there are several significant issues which are not addressed there. Probably the most important is that of intervening, or blocking, variables. Our own experience and intuition strongly suggest that the influence of organizational structure on communication effort is neither direct nor exclusive. There are other factors that affect the amount of time and effort an interaction takes. We have relied on our own experience, and on conversations with many experienced managers, developers, and researchers at the study site, to identify these factors.

Another issue which is not resolved in the literature is a satisfactory way of modeling a process in terms of its individual interactions. In many cases, a process model or definition document will be written in such a way that the required interactions (as defined in section 1.3) are clearly defined. But even when this is the case, it is not clear that the model accurately reflects reality. What rules exist for separating one interaction from another? The breakdown of interactions presented in section 3.3.3 went through a number of iterations until we found a model that both reflected reality and which facilitated the collection of data.

From research questions to variables to research design, the proposed work is supported in the literature. We have extended the current state of the literature not only by combining pieces that have not previously been combined, but also by adding new approaches that were necessary to adequately address the issues of interest.

3. Research Methods

This empirical study examines the role of organizational structure in process communication among software developers. This section explains in detail the methods we employed to investigate this issue. In the subsections which follow, we first present an overview of the research plan and a discussion of our unit of analysis. Then we describe the setting of the study. Finally, the details of data collection, coding, and analysis are presented.

3.1. Overview

Our research design combines qualitative and quantitative methods. There are a number of ways in which such methods have been combined in studies in the literature. The practice adopted for this study is to use qualitative methods to collect data which is then quantified, or coded, into variables which are then analyzed using quantitative methods. Examples of this practice are found in [Sandelowski92, Schilit82, Schneider85].

The data collection procedures used in this study are participant observation [Taylor84] and structured interviews [Lincoln85]. Development documents from the environments under study also provide some of the data [Lincoln85]. As described later, the data gathered from these different sources overlaps, thus providing a way of triangulating [Lincoln85], or cross-checking the accuracy of the data.

After the data was collected, it was coded in such a way that the result is a set of data points, each of which has a set of values corresponding to a set of quantitative research variables. For example, although participant observation in general yields qualitative (non-quantified) data, we used this data to count the number of people present at the observed meeting, to time the different types of interactions that take place, and to determine what type of communication medium was used. These quantified pieces of data constitute values for the research variables.

The data analysis part of the research design is mostly quantitative. The coded data set was analyzed using very simple statistical methods. Histograms were constructed to determine the distributions of single variables, and scatterplots were used to study the relationships between pairs of variables. Various subsets of the data were also viewed in this way in order to gain a deeper understanding of the findings.

3.2. Unit of Analysis

A brief discussion of the unit of analysis is in order. The unit of analysis in this study is the *interaction*. In section 1.3, an interaction was defined as an instance of communication, in which two or more people are explicitly required (by the process they are executing) to share some piece of information. It should be noted that only process-oriented interactions are considered in this study. For example, a document handoff between different sets of developers, a review meeting, and a joint decision on how to proceed are all considered interactions in this context if they are required as part of some defined process step. We would not include, for example, informal (optional) consultations on technical matters between developers, even though this type of communication might be "required" because a developer cannot accomplish a given task adequately without it. Such informal communication is a very important, but we believe separate, area of research.

Most social science research methods assume that the unit of analysis is a person, and methods are described with this assumption, at least implicitly. However, there are some examples in the literature of empirical studies which use a unit of analysis other than a person or group of people. One is Schilit's [Schilit82] study of how workers influence the decisions of their superiors in organizations. The unit of analysis in this study is called an interaction, in this case an attempt by a subordinate to influence a superior in some way. The research variables represented characteristics of such interactions, e.g. method of influence. Like our study, this is an example in which the unit of analysis is a phenomenon, or an event, rather than a person. Other examples of non-human units of analysis can be found in the research literature on group therapy.

There are several ramifications of using this type of unit of analysis. First of all, the "size" of the study cannot be stated in terms of people. The number of people interviewed or observed is not a meaningful size measure since not all people involved in interactions are interviewed, and the same person may be present in a number of observations. The size of the study is *the number of interactions*. Each interaction constitutes one data point, and all of the variables are evaluated with respect to an interaction.

Another possible complication with this unit of analysis is the problem of independence. Any analysis method that attempts to describe a relationship between variables has an underlying requirement that the values (of variables) associated with one data point are not in any way dependent on the values associated with another data point. It can be argued that, since different interactions can involve the same people, they may not be independent. It's not clear, however, that independence has this meaning when the unit of analysis is not a person. It can be argued that the properties of people that are relevant in our context are represented as variables, and thus any dependence between two data points simply means that they share the same values for some variables. In any case, this issue should be taken into account when assessing the results reported in section 4.

3.3. Study Setting

This study took place at IBM Software Solutions Laboratory in Toronto, Canada. The development project studied was DB2, a commercial database system with several versions for different platforms. During the month of June 1994, data was collected from (mostly design and code) reviews in Toronto. Ten reviews were directly observed, which involved about 100 interactions. These observations were followed up with interviews with review participants in November 1994, and in April 1995. The review process was chosen for study because it is well-defined in the DB2 project, it involves a lot of communication between participants, and much of it is observable.

A three-part model of the DB2 development environment was built. The model had several purposes. First, it was used to better understand the DB2 review process and the people involved. Also, it served as a vehicle with which to communicate with developers and others from whom we were collecting information. Finally, it was used as a framework in which to organize the data.

Recall that the issue which motivates this work is the relationship between development organizations and processes. Information flow is one area in which organizational and process issues come together. To reflect this, the model of the DB2 environment is organized in three parts. One part corresponds to the development process under study, one to the organizational structure in which that process is executed, and one to the intersection between the two, which is modeled as a set of interactions. In section 1.3, we defined an interaction, as it is used here, as an instance of communication in which two or more process participants must share some piece of information in order to carry out their process responsibilities. The three parts of the model are described in sections 3.3.1 through 3.3.3.

3.3.1. Process

The work that goes into each release of DB2 is divided into *line items*, each of which corresponds to a single enhancement, or piece of functionality. Work on a line item may involve modification of any number of software components. For each line item, reviews are conducted of each major artifact (requirements, design, code, and test cases). In this study, we observed and measured reviews of all types, but mostly design and code.

The review process consists of the following general steps:

Planning - The Author and Line Item Owner (often the same person) decide who should be asked to review the material. The Author then schedules the review meeting and distributes the material to be reviewed.

Preparation - All Reviewers read and review the material. Some Reviewers write comments on the review material, which they later give to the Author. The Chief Reviewer sometimes checks with each Reviewer before the meeting to make sure they have reviewed the material.

Review Meeting - There are a number of different ways to work through the material during the meeting, and to record the defects. In some cases, the Moderator records all defects raised on Major Defect Forms, which were given to the Author at the end of the meeting. In other reviews, the Moderator does not use the forms, but writes down detailed notes of all defects, questions, and comments made. In still others, the Moderator takes only limited notes and each Reviewer is expected to provide written comments to the Author. In all cases, the Reviewers make a consensus decision about whether a re-review is required. Also, at the end of the meeting, the Moderator fills out most of the Review Summary Form and gives it to the Chief Reviewer.

Rework - The Author performs all the required rework.

Follow up - The Chief Reviewer is responsible for making sure that the rework is reviewed in some manner. This could take place in an informal meeting between the Author, Chief Reviewer, and sometimes the Line Item Owner. In other cases, the Author simply gives the Chief Reviewer the reworked material and the Chief Reviewer reviews it at his or her convenience. After the rework is reviewed, the Chief Reviewer completes the Review Summary Form and submits it to the Release Team.

3.3.2. Organization

The formal DB2 organization has a basic hierarchical structure. First-line managers are of three types. Technical managers manage small teams of programmers who are responsible for maintaining specific collections of software components. Groups of developers reporting to a technical manager may be further divided by component and Task Leaders may be assigned to head each subgroup. Product managers are responsible for managing releases of DB2 products. Teams reporting to product managers coordinate all the activities required to get a release out the door. Support managers manage teams that provide support services, like system test, to all the other teams. There is one second-line manager responsible for all DB2 development.

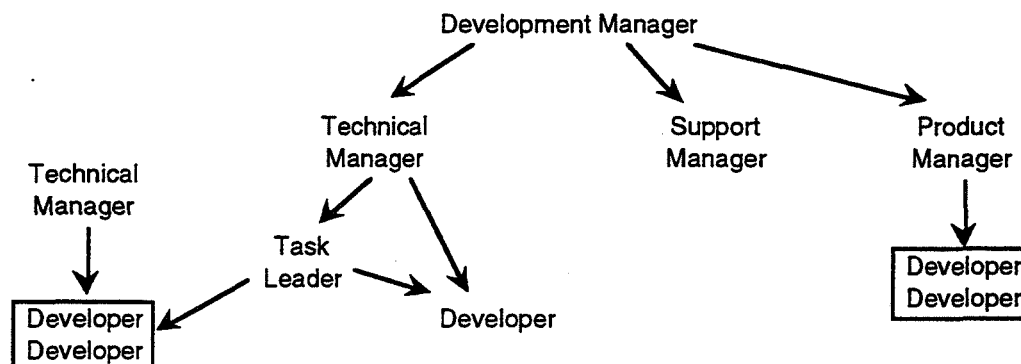


Figure 1. Example reporting relationships.

The part of the three-part model which depicts the organizational structure of the DB2 team consists of a set of simple graphs. Each graph shows a different perspective on the organizational structure. In each, the nodes are the people that constitute the team or are

relevant to the development process in some way. The edges or groupings show different relationships between the members of the organization. One graph (an example appears in Figure 1) shows the reporting relationships, and is derived from the official organization chart. Note that the reporting structure need not be strictly hierarchical, and official relationships other than the traditional manager/employee relationship can be represented (e.g., the "Task Leader"). Another (Figure 2) shows the same organization members linked together according to work patterns. Those people that work together on a regular basis and are familiar with each others' work are linked or grouped. A third graph reflects the physical locations of the members of the organization (Figure 3). People who share offices, corridors, buildings, and sites are grouped with different types of boxes. These graphs are used to measure several properties of the relationships between organizational members (see section 3.5).

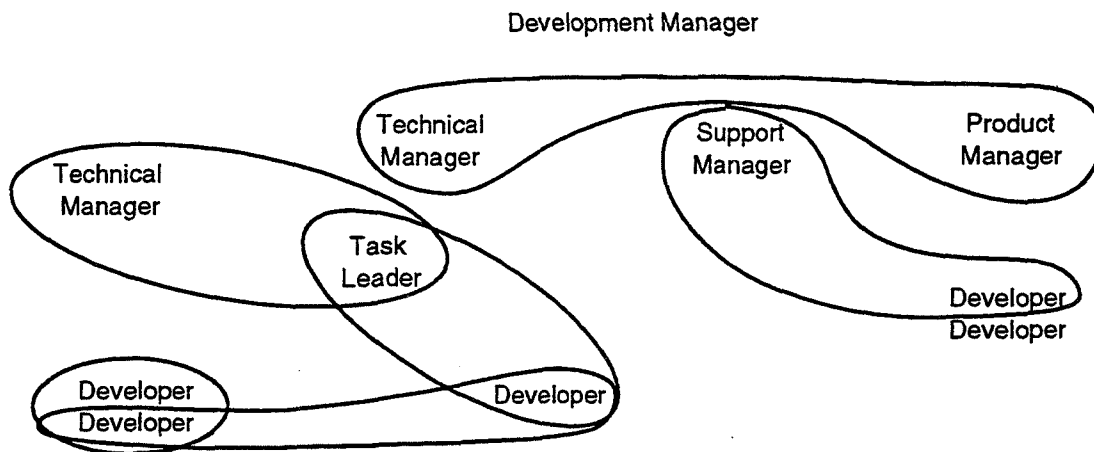


Figure 2. Example working relationships.

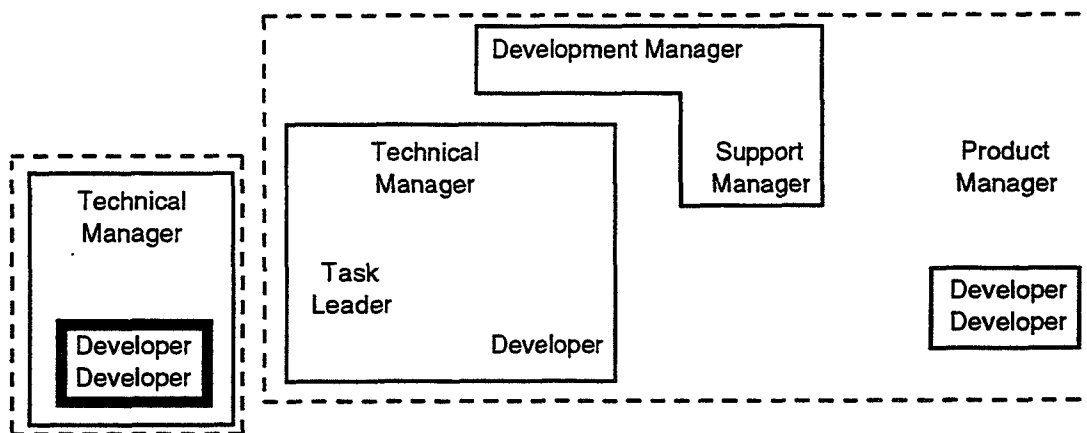


Figure 3. Example physical proximity relationships

3.3.3. Interactions

The third part of the model of the DB2 environment is made up of the types of interactions, or instances of communication, that are both dictated by the defined software development process and that actually take place, between members of the organization. These

interactions constitute the overlap, or relationship, between the DB2 process and organization.

Each interaction has a set of participants. Interactions also have a *mode* of interaction, which restricts which participants interact with which others. If an interaction is *multidirectional*, then the set of participants is considered one large set, and all participants interact with all other participants. Information flows in all directions between all participants. If an interaction is *unidirectional* or *bidirectional*, then the participants are divided into two sets. Participants in one set interact only with those in the other set. In the case of unidirectional interactions, information flows from one set to the other. In bidirectional interactions, information flows in both directions.

Below are the types of interactions we have identified as *potentially* occurring during any review. Type names are meant to describe the information that is shared during the interaction:

choose_participants - the Author and Line Item Owner choose the Reviewers
review_material - the Author gives the material to be reviewed to the Reviewers
preparation_done - the Chief Reviewer asks each Reviewer if they have completed reviewing the material
schedule_meeting - the Author schedules the review meeting at a time convenient to all Reviewers
commented_material - one or more Reviewers give copies of the reviewed material, with their written comments on it, to the Author
comments - the Moderator gives the comments he or she has recorded during the review meeting to the Author
summary_form - the Moderator gives the partially completed Review Summary Form to the Chief Reviewer
summary_form_rework - the Chief Reviewer gives the completed Review Summary Form to the Release Team
questions - Reviewers raise and discuss questions with the Author during the review meeting
defects - Reviewers raise and discuss defects with the Author during the review meeting
discussion - Reviewers and the Author discuss various issues related to the line item during the review meeting
re-review_decision - all Reviewers decide whether or not a re-review is required
rework - the Author, Line Item Owner, and Chief Reviewer review the rework

Not all of the interactions listed above occurred during all reviews. Those that did occur, however, are represented just once for that review. For example, there is only one **questions** interaction for each review meeting. Although a number of questions may have been raised and discussed, they all involve the same set of people, use the same communication media, and refer to the same document (the design or code being reviewed). Thus all of the independent variables have the same values for each question raised. Consequently, for notational and computational convenience, we have modeled the **questions** interaction as a single interaction per review. The same is true for the **defects** and **discussion** interactions.

These identified interactions constitute the unit of analysis for this study, as explained in section 3.2. That is, each data point corresponds to an interaction of one of the types listed above. In addition, the *type* of an interaction (e.g. **defects**, **rework**, etc.) is one of the variables we shall use in the analysis of the data.

3.4. Data Collection

The data for this study were collected during an initial visit to IBM in Toronto in June 1994, two follow-up visits in November 1994 and April 1995, and several email communications in between the visits. The data collection procedures included gathering of official documents, participant observation [Taylor84], and structured interviews [Lincoln85]. The observations and interviews were guided by a number of forms, or instruments. All of these procedures and instruments are discussed in the following sections.

Checklist for observing reviews:						
Release:						
Line Item:						
Review type:						
Date:						
Author(s):						
Moderator:						
Chief Reviewer:						
Reviewers:						
Total prep:						
Meeting length:						
# participants:						
Amount reviewed:						
Is this a re-review?						
Is a re-review planned?						
Defects:	FPFS	HLD	LLD	CODE	TPLAN	TCASES
Major						
Minor						
Time to:						
read (1 person on average):						
fill out summary form:						
questions:						
discuss errors:						
other discussion:						
Log (on other side)						
Categories of time:						
Questions (Q)						
Error discussion (E)						
Other discussion (D)						
Summary form (SUM)						
Administration (A)						
Notes:						

Figure 4. The observation checklist

3.4.1. Documents

The official documents of an organization are valuable sources of information because they are relatively available, stable, rich, and non-reactive, at least in comparison to human data sources [Lincoln85]. The model of the DB2 environment (described in section 3.3) relied initially on two confidential IBM documents, a review process document and the official organization chart.

Other documents which provided data later were copies of the *Review Summary Forms* for each review that was observed. Most of the information on these forms had already been collected during the observations of the reviews, so the forms served as a validation (triangulation) instrument.

3.4.2. Observations

Participant observation, as defined in [Taylor84], refers to "research that involves social interaction between the researcher and informants in the milieu of the latter, during which data are systematically and unobtrusively collected." Examples of studies based on participant observation are found in [Barley90, Perry94, Sandelowski92, Sullivan85]. In these examples, observations were conducted in the subjects' workplaces, homes, and therapists' offices. The idea, in all these cases, was to capture firsthand behaviors and interactions that might not be noticed otherwise.

Much of the data for this study was collected during direct observation of 10 reviews of DB2 line items in June 1994. Figure 4 shows a copy of the form, called the observation checklist, that was filled out by the observer for each review. Most of the administrative information on the form was provided in the announcement of the review, the Review Summary Form, or by the participants during or after the review meeting.

During the course of the review, each separate discussion was timed. The beginning and ending times, the participants, and the type of each discussion were listed on the back of the observation checklist for the review. A discussion constituted the raising of a defect (E) if it ended in the Moderator making an entry on a Major Defect Form or the list of defects he or she was keeping. A question (Q) was a discussion that did not end in an entry on the defect list, and additionally had begun with a question from one of the Reviewers. Other discussions (D) were those that neither began with a question nor ended with the recording of a defect. Some time was spent filling out the summary form (SUM), for example the reporting of preparation time for each Reviewer. Finally, a small amount of time in each review was spent in administrative tasks (A), for example connecting with remote Reviewers via phone. Totals for these various categories of time were recorded on the observation checklist.

3.4.3. Interviews

In [Lincoln85], a structured, or "focused", interview is described as one in which "the questions are in the hands of the interviewer and the response rests with the interviewee", as opposed to an unstructured interview in which the interviewee is the source of both questions and answers. The interviews conducted for the pilot study were structured interviews because each interview started with a specific set of questions, the answers to which were the objective of the interview. Examples of studies based on interviews can be found in [Rank92, Schilit82, Schneider85].

Interview Guide for John Doe, MM/DD/YY

1. How much of your prep time is spent filling out the defect form?
Recording minor defects?
2. How much time is taken up by scheduling and distributing materials?
3. How long was the followup meeting?
4. Do you work much with the other participants, aside from reviews?

Figure 5. Example interview guide.

The initial interviews were conducted within a few days of each DB2 review. Other interviews took place in November 1994, and April 1995. One goal of each interview was to elicit information about interactions that were part of the review process but that took place outside the review meeting (and thus were not observed). The interviews also served to clarify some interactions that went on during the meeting, and to triangulate data that had been collected during the meeting. Before each interview, the interviewer constructed an interview form, or guide [Taylor84], which included questions meant to elicit the information sought in that interview. These forms were not shown to the interviewee, but were used as a guide and for recording answers and comments. Figure 5 shows an example of such a guide. For each review, at least one Author and, with one exception, the Chief Reviewer was interviewed. As well, in most cases, several other Reviewers were interviewed.

3.5. Measures

This section describes the procedures used to transform the data collected, as described in the last section, into quantitative variables. First the list of variables is presented, then the details of how the information from documents, observations, and interviews is coded to evaluate these variables.

3.5.1. Variables

The variables chosen for analysis, listed in Table 1, fall into three categories. First is the dependent variable, *Communication Effort*. Secondly, there is a set of independent variables which represent the issues of interest for this study, i.e., organizational structure. Several different measures have been chosen which capture different relevant properties of organizational structure. Finally, there is a large set of variables which are believed to have an effect on communication effort, but which are not the primary concern of this study. If these variables are not taken into account, they threaten to confound the results by hiding the effects of the organizational structure variables.

The dependent variable, labelled **CE** (for Communication Effort), is the amount of effort, in person-minutes, expended to complete an interaction. This is a positive, continuous, ratio-scaled variable and is calculated for each interaction.

There are four organizational structure variables. They are measured in terms of the set of participants in an interaction. The first two, **XOD** and **MOD**, are closely related. They are both based on *Organizational Distance*, which quantifies the degree of management structure between two members of the organization. Using a graph as shown in Figure 1, the shortest path between each pair of interaction participants is calculated. If a shortest path value is 4 or less, then this value is the Organizational Distance between that pair of participants. If it is more than 4, then the Organizational Distance for the pair is 5.

Note that this definition of Organizational Distance does not assume that the management structure is strictly hierarchical. Choosing the shortest path between each pair of participants allows for the possibility that more than one path exists, as might be the case in a matrix organization. It should also be noted that the links representing management relationships are not weighted in any way. One enhancement of this measure would be the addition of weights to differentiate different types of reporting relationships.

The higher values of Organizational Distance have been combined into one category for two reasons. First of all, the data showed that most pairs of interaction participants had an Organizational Distance of 4 or less. Also, it was impossible to calculate Organizational Distance accurately between some very distant pairs of participants. For example, reviews sometimes included a Reviewer from another IBM company. In these cases, the management links to the outside Reviewer were not well defined. Any pair that included that participant would then have an Organizational Distance of 5.

XOD and **MOD** are both aggregate measures of Organizational Distance. They differ in the way that Organizational Distance values for individual pairs of participants are aggregated into a value for an entire set of participants. **XOD** is defined as the maximum Organizational Distance, and **MOD** is the median Organizational Distance, among all pairs of participants in an interaction. Therefore, **XOD** would be high for those interactions in which even just one participant is organizationally distant from the others. **MOD** would be high only for those interactions in which at many of the participants are organizationally distant. The median was chosen for **MOD** because the shortest path values for each pair of participants are ordinal, not interval, and so the mean is not appropriate.

The two other organizational variables are *Familiarity* (**Fam**) and *Physical Distance* (**Phys**). They are also based on pairs of interaction participants, and rely on graphs like those shown in Figures 2 and 3. They are both ordinal. Their levels are shown in Table 1. Familiarity reflects the degree to which the participants in an interaction work together or have worked together outside the review, and thus presumably share common internal representations of the work being done. The familiarity measure also attempts to capture the important informal networks. Physical Distance reflects the number of physical boundaries (walls, buildings, cities) between the interaction participants.

Variable Name	Label	Levels		Data Source(s)
		Code	Meaning	
Communication Effort (ratio)	CE	-	amount of effort, in person-minutes, expended to complete the interaction	observations; 6/94 and 5/95 interviews; Review Summary Forms
Organizational Distance (ordinal)	MOD	1-4	value of the median shortest path between participants in management structure if it is between 1 and 4	organization charts; observations; 6/94 and 5/95 interviews
		5	value of the median shortest path between participants in management structure if it is greater than 4	
	XOD	1-4	value of the maximum shortest path between participants in management structure if it is between 1 and 4	organization charts; observations; 6/94 and 5/95 interviews
		5	value of the maximum shortest path between participants in management structure if it is greater than 4	
Familiarity (ordinal)	Fam	1 2 3 4	pairs of participants who are familiar with each others' work ≤ 10% 10% < pairs of participants who are familiar with each others' work ≤ 20% 20% < pairs of participants who are familiar with each others' work ≤ 50% pairs of participants who are familiar with each others' work > 50%	6/94 and 5/95 interviews
Physical Distance (ordinal)	Phys	1 2 3 4	all participants in same office all participants on same corridor, but not all in same office all participants in Toronto but not all on same corridor at least one pair of participants at different sites	observations; online directory; 5/95 interviews
Number of Participants (absolute)	N	-	number of people participating in an interaction	observations; Review Summary Forms
Skill Level (ordinal)	K	1 2 3	low medium high	11/94 interviews
Request Medium (nominal)	Mr	0 1 2	no request made verbal request electronic request	observation; 5/95 interviews
Preparation Medium (nominal)	Mp	1 2 3 4 5	a written, paper form verbal, with no notes shared a brief written message a structured written document an unstructured written document	observation; 5/95 interviews
Transfer Medium (nominal)	Mt	1 2 3 4 5 6	face-to-face meeting conference call video conference electronic transfer paper 2-way phone call	observation; 5/95 interviews

Variable Name	Label	Levels		Data Source(s)
		Code	Meaning	
Information size (ordinal)	Size	1	very small	observation; 6/94 interviews; Review Summary Forms
		2	≤ 3 pages	
		3	< 1 KLOC or 4-9 pages	
		4	1-2 KLOC or 10-25 pages	
		5	> 2 KLOC or > 25 pages	
Technicality (ordinal)	Tech	1	non-technical	dictated by type of interaction
		2	mixed	
		3	technical	
Complexity (ordinal)	Comp	1	very easy	6/94 interviews
		2	easier than average	
		3	average	
		4	more difficult than average	
		5	very difficult	
Structure (ordinal)	Struct	1	highly structured	observation
		2	mixed	
		3	unstructured	
Information Use (nominal)	Use	1	informational	dictated by type of interaction
		2	decisional	
		3	directional	
		4	functional	
Interaction Type	Type		see Section 3.3.3	

Table 1. Variables used in this study

The set of blocking variables is large. The first is the size of the set of interaction participants. This variable, labelled **N**, is simply the number of people who expend effort in an interaction.

Another blocking variable is skill level, **K**. This variable reflects the level of skill possessed by the person most responsible for an activity, relative to the skills required to complete the activity. The assumption is that the more skilled a person is, the less he or she will depend on other people, and consequently the less time he or she will spend in communication. Many very simple interactions have a high (3) value for **K**, because such interactions do not require much skill. For other interactions, which are more technical in nature, **K** is set equal to the skill level of the Chief Reviewer.

We also wish to block according to the type of communication media used in an interaction. Three different parts of an interaction have been identified that require (potentially different) communication media. The first is the medium used to request information. Since many interactions involve unsolicited information, there is no request and thus no medium for this purpose. In the interactions we studied, when such a request was made, it was made either verbally or via email. Thus this variable, labelled **Mr**, is coded as either a 0 (n/a), 1 (verbal), or 2 (email).

The second part of an interaction which is affected by the choice of communication medium is the preparation of the information to be shared. In the interactions studied, information was prepared in one of five different ways. Some information required simply the completion of a paper form. Other information was to be shared verbally, and required only that the sender prepare his or her thoughts. In this case, written notes might be prepared, but were not shared with other participants in the interaction. The third preparation medium is the writing of a brief, informal message. Fourth is the preparation of a formal document which follows a defined format and structure. Finally, some interactions required the information in the form of an unstructured document. The

blocking variable **Mp**, the medium used to prepare the information to be shared, is coded as a 1 (form), 2 (verbal), 3 (message), 4 (structured document), or 5 (unstructured document).

Finally, an interaction requires a communication medium to transfer the information between participants. The "transfer" media that were used in the interactions studied were face-to-face meetings, conference calls, video conferences, electronic transfer (email, ftp, etc.), paper, and normal phone calls. These values of the variable, labelled **Mt** are coded with numbers 1 through 6, respectively.

The last few blocking variables concern properties of the information that is to be shared in an interaction. The first of these variables is the amount of information. The information in most interactions studied was represented by the material that was being reviewed. In some cases, this material was code, measured in LOC, and in others it was a design document, measured in pages. In order to collapse these two "size" measures, we have created an ordinal scale, shown in Table 1, under "Information Size". The first level refers to interactions where the information shared is very simple, e.g. the answer to a "yes or no" question. These interactions are normally part of the managerial tasks surrounding a review. The second level is also used in some managerial interactions that involve a bit more information, e.g. a review summary form, and also reviews of small design documents. The last three levels each correspond to both a number of lines of code and a number of pages. The two definitions of each level were considered roughly equivalent by a number of experienced reviewers. The boundaries between these levels were chosen based on the data, which naturally fell into these groups. The amount of information in an interaction is labelled **Size**.

The second information characteristic is the degree of technicality of the information. This variable, labelled **Tech**, is coded as 1 for non-technical (managerial or logistical) information, 2 for information which is mixed, and 3 for purely technical information.

Information complexity, **Comp**, is coded on a five-point subjective scale, based on the answers to questions put to developers about the comparative complexity of the material being reviewed. **Comp** ranges from very easy (1) to very hard (5). This variable is meant to capture how much difficulty interaction participants would have in understanding the information. Information in managerial or logistical interactions is generally not very complex (usually 1). Review materials, however, vary over the entire range.

The degree of structure that the information exhibits, labelled **Struct**, reflects whether or not the information follows some predefined format or is in some language more precise than English. Source code, for example, is highly structured (1), as is information on a form. Questions and discussion are unstructured (3). Design documents, which are written with a predefined template, are in between (2).

The use to which it is to be put after the interaction is another characteristic of information. That is, we want to record the purpose of the interaction and the reason the information needs to be shared. This variable also gives some indication of the *importance* of the information. This variable, labelled **Use**, has the value 1 if the purpose of the interaction is general information, and it is not clear what specific activities or decisions will be affected by this information in the future. A value of 2 indicates that the information will be used to help make a decision. Some information is used to influence how and which activities are performed (3). This is often logistical information, for example a deadline. Finally, information can be used as input to an activity (4), for example a design document as input to the implementation activity.

Finally, we will use the type of interaction, **Type**, as a variable later in our analysis. The type of an interaction is related to the step of the process it is part of and the information involved. The types of interactions for this study were identified during construction of the three-part model presented in section 3.3. The list of interaction types is presented in section 3.3.3.

3.5.2. Coding

Recall that each review has associated with it a set of interactions, each of which has a value for every variable. Thus, there is an instance of each variable for each interaction in each review. The values of some independent variables are completely determined by the type of interaction. That is, some variables have the same value for every interaction of a certain type, regardless of which review it is part of. The values of these variables are dictated by the way that interactions have been modeled. For other variables and other interactions, the values vary over different reviews. This information is summarized in Table 1.

The dependent variable, **CE**, was coded by combining several pieces of data, depending on the interaction. Recall that **CE** for an interaction is defined as the total amount of effort expended on that interaction, from the initial request for information through the reading (or otherwise digesting) of the information.

The effort for many interaction types (e.g. **schedule_meeting**, **choose_participants** and **preparation_done**) is straightforwardly gathered from interviews. In many cases, the effort information gathered in interviews reflects the amount of time just one of the participants spent in the interaction, so the value must be multiplied by the number of participants.

The effort for the interactions which take place during the review meeting is actually observed and timed. These values must also be multiplied by the number of people present at the meeting. This is not always straightforward, as it was common for reviewers to come and go during the course of the review meeting. Some of these interactions also include some of the preparation time (e.g. reviewers prepare the defects to be raised at the meeting ahead of time), so that is included in the calculation of **CE**.

Values of the organizational variables, Organizational Distance (**XOD** and **MOD**), Familiarity, and Physical Distance, are calculated directly from the graphs that make up the organizational part of the model described in section 3.3.2. The scales used for these variables are shown in Table 1. These scales were derived from the data itself.

Most of the information used to evaluate the blocking variables was collected during interviews. Some blocking variables, however, were evaluated a priori according to the type of interaction. For example, some interaction types always involve technical information while others are concerned with purely managerial or logistical information. So the value of the variable **T** is constant for each type of interaction, regardless of any characteristic of individual reviews.

3.6. Data Analysis

The data to be analyzed consisted of 100 data points, each corresponding to a single interaction. Associated with each data point were values for each of the independent,

dependent, and blocking variables. In addition, we recorded for each data point what type of interaction it corresponded to, which review that interaction was a part of, and the nature of the material being reviewed (code, design, etc.).

The data analysis involved the construction of histograms to display the distributions of individual variables, and scatterplots to illustrate the relationships between pairs of variables. Blocking variables were used in a limited way to explore whether or not relationships between variables held under different conditions. Part of the analysis also involved blocking the data by interaction type.

Our analysis method basically consisted of creating subsets of data based on the values of one or more variables, then creating histograms and scatterplots based on those subsets. The subsets of interactions that we analyzed are:

- the entire set of interactions
- high effort interactions (**CE**>250 and **CE**>500)
- technical interactions which take place during the review meeting (**questions, defects, and discussion**).
- by technicality
- by complexity
- by degree of structure
- by size of information
- by skill level
- by number of participants
- by interaction type
- by line item
- by combinations of the organizational variables (e.g. low Physical Distance and high **MOD**)

For each of these subsets, a histogram showing the distribution of **CE** was generated, as well as scatterplots showing the relationships between **CE** and each organizational variable (**MOD**, **XOD**, Physical Distance, and Familiarity). To test these relationships, Spearman correlation coefficients were calculated. In addition, the distributions of other variables were analyzed for some of the subsets. For example, we looked at the distribution of interaction types among the high-effort interactions. Also, for both the data set as a whole and for the high-effort interactions, we studied the distributions of all variables. Another two-variable relationships that we explored with scatterplots is the relationship between **CE** and the number of participants (**N**). For this relationship, we grouped the data by line item to see which line items required more Communication Effort overall, and which required more effort per participant. We also ran ANOVAs (Analysis of Variance tests) on some combinations of variables for some subsets, but there was not enough data to yield meaningful results. Mann-Whitney tests were also used to test some special hypotheses about combined effects of Organizational Distance. The strongest and most interesting of our findings are presented in the next section.

4. Results

The results of our study are presented in four subsections. First, we give an overall characterization of the data collected by looking at each variable in isolation. Then we present some findings concerning the relationships between the dependent variable (Communication Effort) and the various organizational independent variables. We call these relationships "global" because they hold in the data set as a whole. In section 4.3, we

examine those interactions that required the most Communication Effort more closely. Finally, in section 4.4, we divide the data by type of interaction to see what patterns emerge for different types.

4.1 Data Characterization

We begin by characterizing the data collected. In particular, we will examine the distributions of values of the variables studied. First, as can be seen in Figure 6, the distribution of the dependent variable, Communication Effort, is highly skewed towards the low end. The box plot at the top shows another view of this distribution. The box itself is bounded by the 25th and 75th quantiles. The diamond indicates the mean of the data. 90% of the interactions had a **CE** of less than 600 person-minutes. The maximum amount of effort any interaction required was 1919 person-minutes, and the minimum was 3 person-minutes. The median was 38 and the mean was about 190.

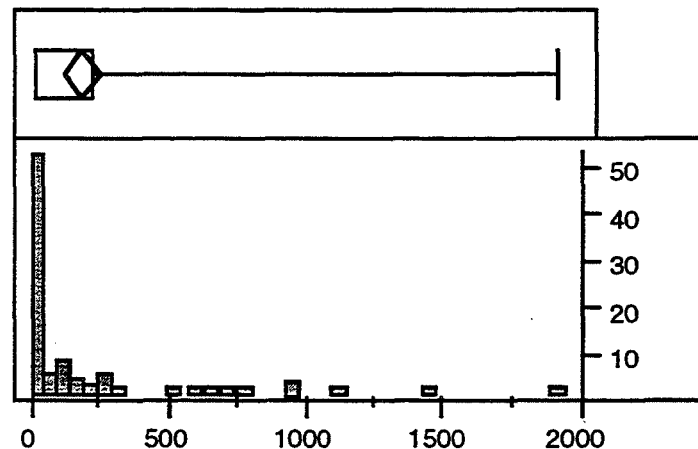


Figure 6. Distribution of Communication Effort (in person-minutes) over all 100 interactions

Level	MOD		XOD	
	Count	Cum %	Count	Cum %
1	32	32	23	23
2	29	61	1	24
3	6	67	0	24
4	27	94	33	57
5	6	100	43	100

Table 2. Frequency table for median (MOD) and maximum (XOD) Organizational Distance

It is also useful to look at the distribution of the independent variables. Table 2 shows the numbers and cumulative percentages of data points at each level of **MOD** and **XOD** (recall that there are exactly 100 data points, so simple percentages are not shown). About 60% of the interactions had a median Organizational Distance (**MOD**) of 2 or less, while more than three quarters had a maximum Organizational Distance (**XOD**) of 4 or higher. If we look at **MOD** and **XOD** together, as in Table 3, we see that most of the data falls into three categories:

- 24% of the interactions have all participants organizationally close (low **MOD**, low **XOD**)
- 37% of the interactions have most of the participants organizationally close, but a few organizationally distant (low **MOD**, high **XOD**)
- 33% of the interactions have most of the participants organizationally distant (high **MOD**, high **XOD**)

We will be referring to these categories later as we examine the differences between them.

MOD \ XOD	1	2	4	5
1	23	0	8	1
2	0	1	16	12
3	0	0	0	6
4	0	0	9	18
5	0	0	0	6

Table 3. Frequency of values for median and maximum Organizational Distance

Figures 7 and 8 show the distributions of Familiarity and Physical Distance. The interactions tend to have low Familiarity (75% with 2 or less) and high Physical Distance (83% with 3 or more).

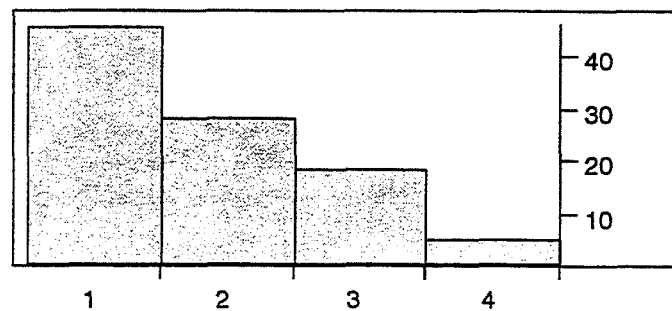


Figure 7. Distribution of Familiarity over all 100 interactions

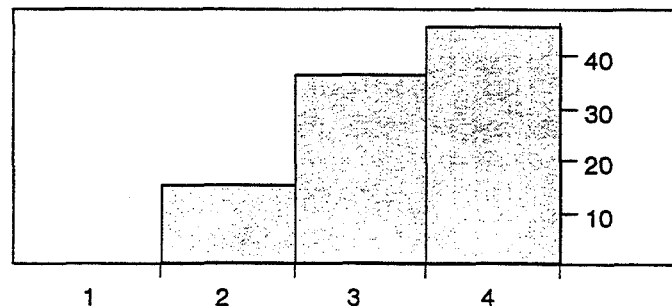


Figure 8. Distribution of Physical Distance over all 100 interactions

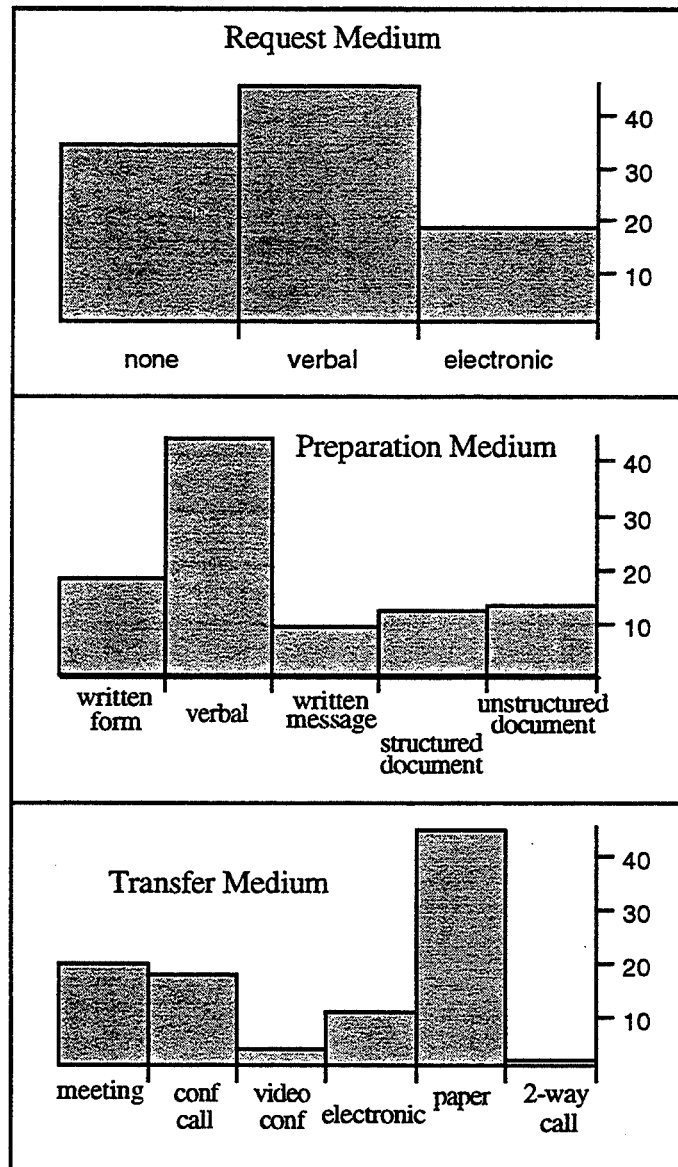


Figure 9. The different communication media used in all 100 interactions.

It is also useful to take a quick look at the characteristics of the data set with respect to the blocking variables. The distributions of the communication media variables are shown in Figure 9. Most interactions either began with a verbal request (46%), or no request at all (35%). The distribution of **Mp** shows that, in many interactions (44%), the information was prepared to be shared verbally. However, each of the other types of information preparation (written forms, messages, and documents) were used in 10-20% of the interactions. The medium most commonly used to actually transfer the information was paper (45%), although face-to-face meetings (20%) and conference calls (18%) were also well represented, along with email (11%).

All of the interactions involved either technical or non-technical information (no mixed), with about 60% of them technical. The information in most of the interactions (53%) was considered less complex than average, although a third were considered slightly more complex than average. The data was fairly evenly divided among interactions involving

structured, unstructured, and mixed information. All different sizes of information were represented, although 50% of the interactions involved small amounts of information (3 pages or less). Almost half of the interactions involved information of a functional nature ($U=4$), 19% was directional, 5% was decisional, and 29% was informational. These distributions are shown in Figure 10.

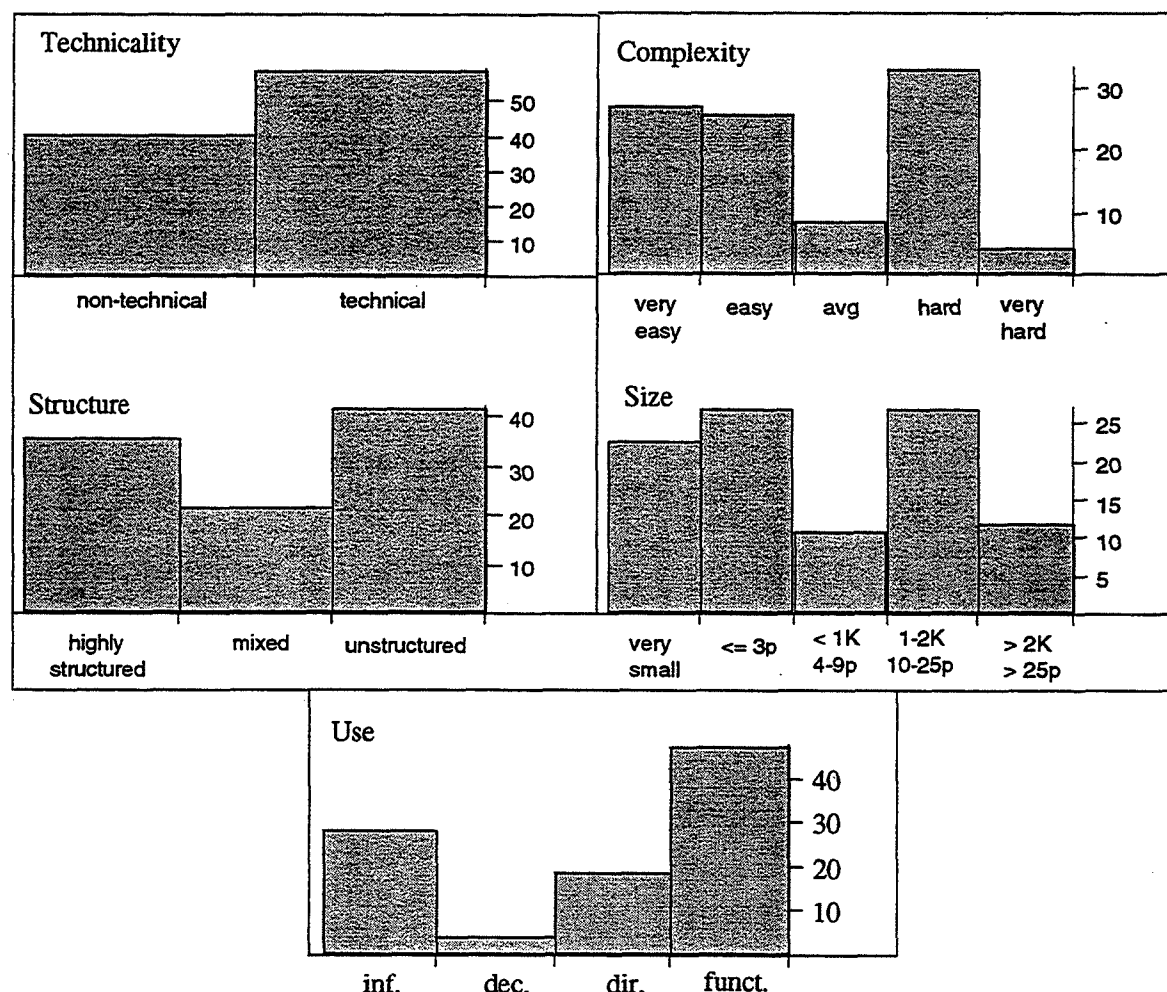


Figure 10. Distributions of blocking variables over all 100 interactions.

4.2 Global Relationships

Next, we want to look at the overall relationship between the dependent variable, Communication Effort, and each of the independent variables in turn. Figure 11 shows two scatterplots, each with Communication Effort on the vertical axis, and one of the two versions of the Organizational Distance variable on the horizontal axis. A boxplot is also shown for each level of each independent variable. The top and bottom boundaries of the boxes indicate the 75th and 25th quantiles. The median and the 90th and 10th quantiles are also shown as short horizontal lines (the median and 10th quantiles are not really visible on most boxes). The width of each box reflects the number of data points in that level.

From Figure 11, we can observe that the highest-effort interactions are those with a relatively low median Organizational Distance (**MOD**) and relatively high maximum Organizational Distance (**XOD**). This is the second category described above, in the discussion of the distributions of **MOD** and **XOD**. This observation implies that groups require more effort to communicate when they include a few (but not too many) members who are organizationally distant from the others. Less effort is required when the group is composed of all organizationally close members (low **MOD** and low **XOD**), or all or nearly all organizationally distant members (high **MOD** and high **XOD**). We tested the statistical strength of this result by calculating the Mann-Whitney U statistic. This is a non-parametric test meant to indicate whether or not two independent samples exhibit the same distribution with respect to the dependent variable (**CE**). In this case, the two groups were those interactions falling into the high **XOD**/low **MOD** category, and those which did not. The test yielded a significant value, even at the $p < .01$ significance level.

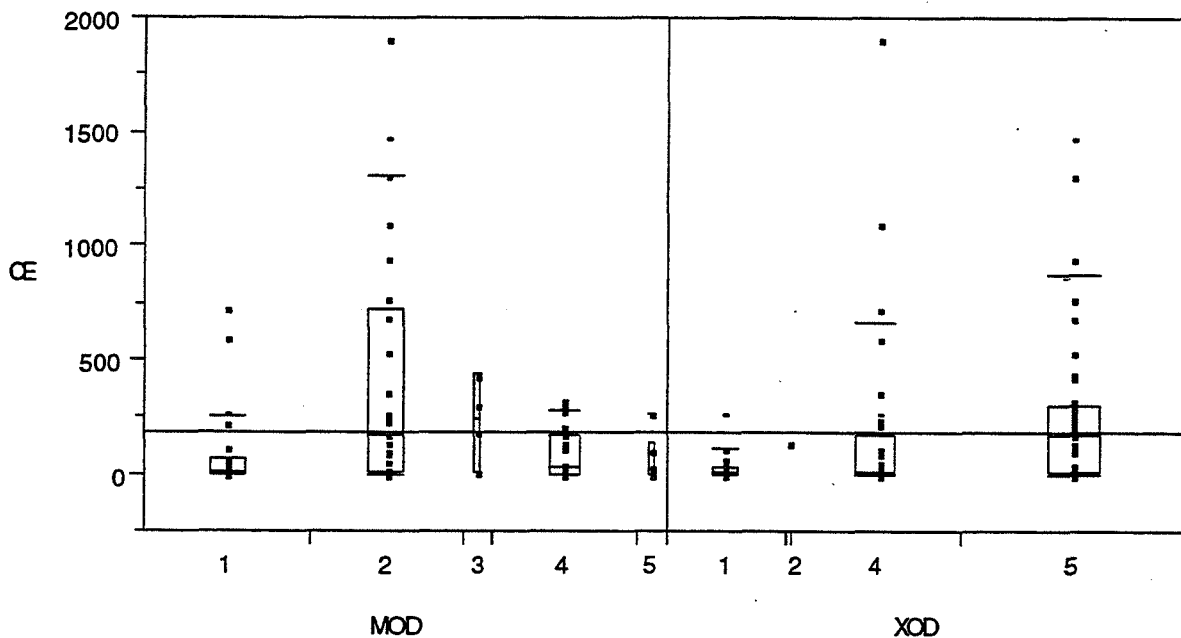


Figure 11. Communication Effort plotted against median Organizational Distance (**MOD**) and maximum Organizational Distance (**XOD**)

We investigated this interesting result about Organizational Distance in more detail by partitioning the data by values of the different blocking variables, and then performing the same Mann-Whitney test on each partition. Again, this test was performed to determine if interactions in the high **XOD**/low **MOD** category exhibited significantly higher levels of **CE** than other interactions. The test was run using both normalized and unnormalized **CE** values for the dependent variable. The results are summarized in Table 4. There are some values of some independent variables that are not used to restrict the data set because the resulting subsets were too small or too homogeneous to yield meaningful results. The values in the table are the " p " values, which indicate, in each case, the probability that the difference in **CE** between interactions with high **XOD**/low **MOD** and other interactions is due to chance. In other words, a low value for p indicates a significant difference in **CE**. Generally, a value of .05 or less is considered significant.

Data set restricted to those interactions with:	Probability that the difference in unnormalized CE is due to chance	Probability that the difference in normalized CE is due to chance
Skill Level = medium	0.0018	0.03
Skill Level = high	0.02	0.83
Use = informational	0.02	0.07
Use = directional	0.13	0.74
Use = functional	0.33	0.96
Size = very small	0.59	0.4
Size <= 3 pages	0.79	0.79
Size = <1KLOC or 4-9p	0.24	0.64
Size = 1-2KLOC or 10-25p	0.04	0.06
Size > 2KLOC or 25pages	0.01	0.03
Structure = highly	0.39	0.33
Structure = mixed	0.84	0.25
Structure = unstructured	0.004	0.03
Complexity = very easy	0.007	0.29
Complexity = easy	0.87	0.87
Complexity = difficult	0.001	0.07
Technicality = non-technical	0.87	0.004
Technicality = technical	0.002	0.04
Mt = face to face	0.29	0.64
Mt = conference call	0.19	0.26
Mt = electronic	1.0	0.93
Mt = paper	0.1	0.1
Mp = paper form	1.0	1.0
Mp = verbal	0.22	0.28
Mp = written message	0.5	0.75
Mp = structured document	0.67	0.67
Mp = unstructured doc.	0.1	0.1
Mr = no request	0.04	0.32
Mr = verbal request	0.19	0.24
Mr = electronic request	0.11	0.17
Phys = 3	0.42	0.69
Phys = 4	0.09	0.58
Fam = 1	0.0004	0.45
Fam = 2	0.2	0.06
Fam = 3	0.02*	0.06

Table 4. p values for the Mann-Whitney U test, comparing **CE** values of interactions with high **XOD**/low **MOD** with other interactions, with the data restricted by the values of other independent variables. Significant values are highlighted.

(*) When the data is restricted by **Fam** = 3, the difference in **CE** is significant but in the opposite direction than expected.

Surprisingly, the results of the test were not significant for many of the data subsets. It was significant for medium skill level, and for unnormalized **CE** at the high skill level. It was also significant, at least for unnormalized **CE**, for interactions involving large amounts of information (the two highest levels of **Size**) but not smaller amounts. The difference was significant for informational interactions, but not for directional or functional (levels of **Use**). Significance was found for interactions involving unstructured information, but not mixed or highly structured (**Struct**). Significance (in unnormalized **CE**) was also found

for two different levels of complexity, "very easy" and "more difficult than average", but not the others. Significance also held for technical interactions, but not administrative ones. Significance was not found for any individual levels of any of the communication media variables, with one exception, nor for any levels of Physical Distance. Partitioning the data by levels of the Familiarity variable produced some interesting results. The Mann Whitney test found a significant difference in unnormalized **CE** for interactions with **Fam** equal to 1, but not 2. For interactions with **Fam** equal to 3, the test was significant, but in the opposite direction. That is, in this subset, interactions in the high **XOD**/low **MOD** category exhibited significantly *lower* levels of unnormalized **CE** than other interactions.

This set of results is difficult to interpret. In general, interactions which have high **XOD** and low **MOD** will require more communication effort. However, the effect of Organizational Distance may be overshadowed by the effect of size, use, degree of structure, complexity, or technicality.

In Figure 12, Communication Effort is plotted against the two remaining independent variables, Familiarity (**Fam**) and Physical Distance (**Phys**). It appears that high effort is associated with low Familiarity and with high Physical Distance (the latter observation being the strongest). However, it must be noted that most interactions have low Familiarity and high Physical Distance.

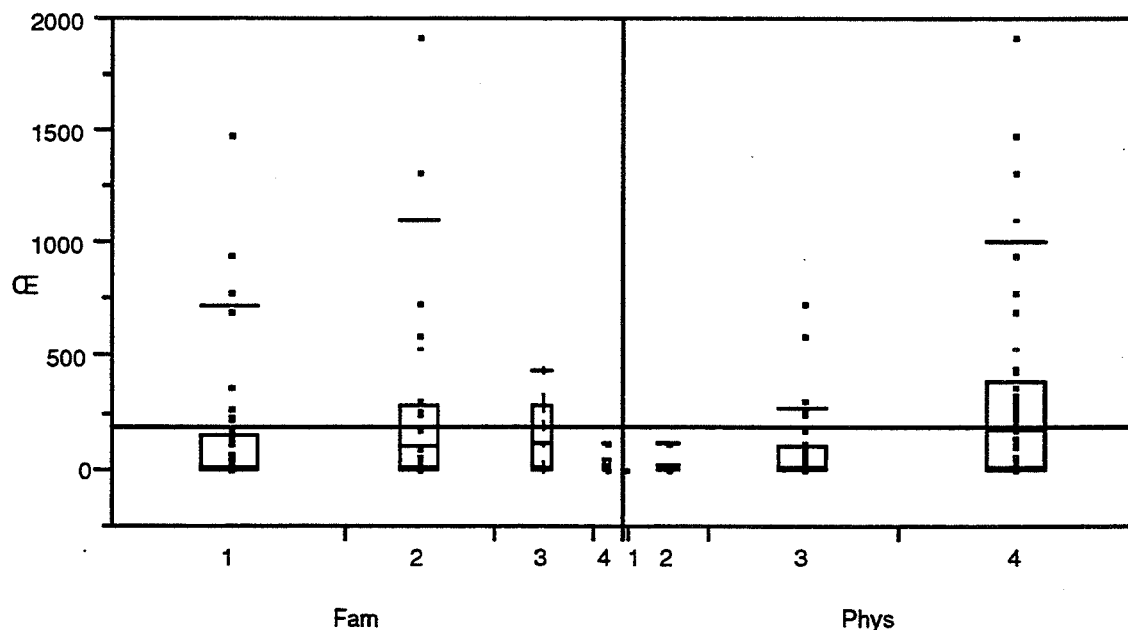


Figure 12. Communication Effort plotted against Familiarity (**Fam**) and Physical Distance (**Phys**)

The Spearman correlation coefficients, which reflect the strength of the relationships between each independent variable and the dependent variable, are shown in Table 5.

MOD	XOD	Fam	Phys
0.09	0.4	0.14	0.5

Table 5. Spearman rho (ρ) coefficients comparing each independent variable to the dependent variable, **CE**.

4.3 High Effort Interactions

In order to investigate all of these possible relationships, we have examined in more detail the subset of interactions which were effort-intensive. In particular, we have chosen the 11 highest-effort interactions, all of which required a Communication Effort greater than 500 person-minutes, and compared the characteristics of this subset to the distributions of the entire subset, described above. The first observation is that **CE** is more evenly distributed in this subset, as can be seen in Figure 13.

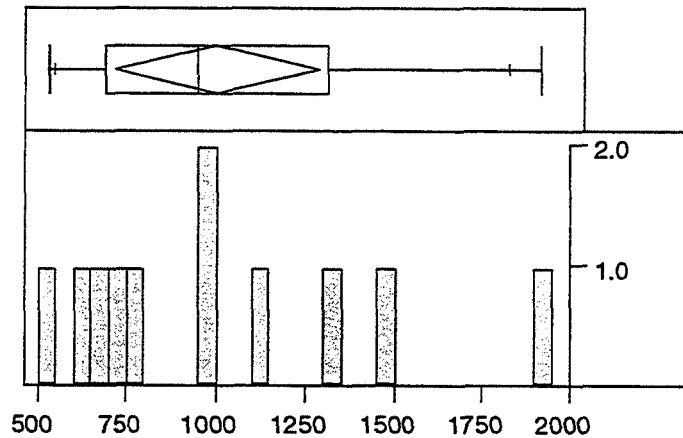


Figure 13. Distribution of Communication Effort over the 11 highest-effort interactions. The y axis is the number of data points.

It should also be noted that the product development team that was studied was divided into two subteams. Each subteam was developing a version of DB2 for a different hardware platform. All of the high-effort interactions took place during reviews conducted by just one of the teams. Most of the high-effort interactions were also either of type **defects** or **questions**.

MOD \ XOD	XOD	
	4	5
1	2	0
2	2	7

Table 6. Frequency of values for median and maximum Organizational Distance for the 11 highest-effort interactions

In looking at the distributions of Organizational Distance in this subset, we noticed that none of the high-effort interactions had a **MOD** more than 2, and none had a **XOD** less than 4. In fact, all of the interactions in this high-effort subset belong to the second category (low **MOD**/high **XOD**) described above, as shown in Table 6.

Also in this subset, we see the same pattern in the Familiarity and Physical Distance variables (Figure 14). That is, interactions tend to have low Familiarity and high Physical Distance both in the data set in general and in the high-effort subset. However, this trend is accentuated in the subset, where none of the interactions have Familiarity more than 2 (as compared to 25% in the whole data set). Similarly, 80% of the high-effort interactions have a Physical Distance of 4, the highest level of this variable.

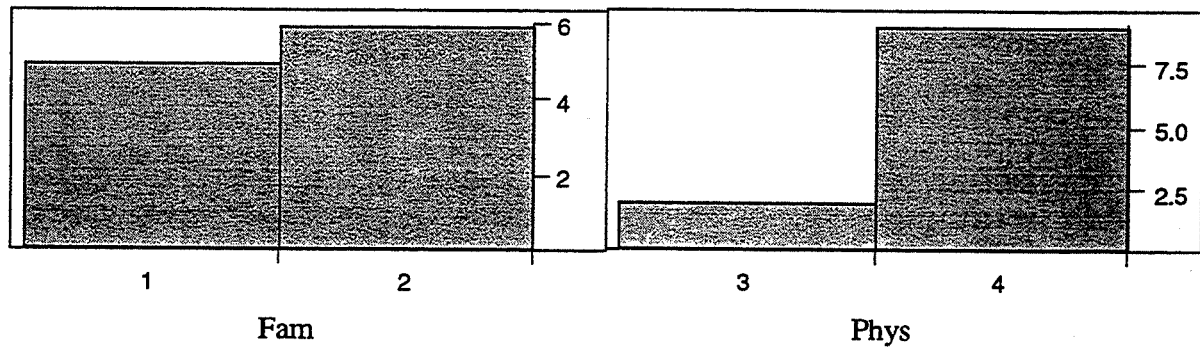


Figure 14. Distributions of Familiarity and Physical Distance among 11 highest effort interactions.

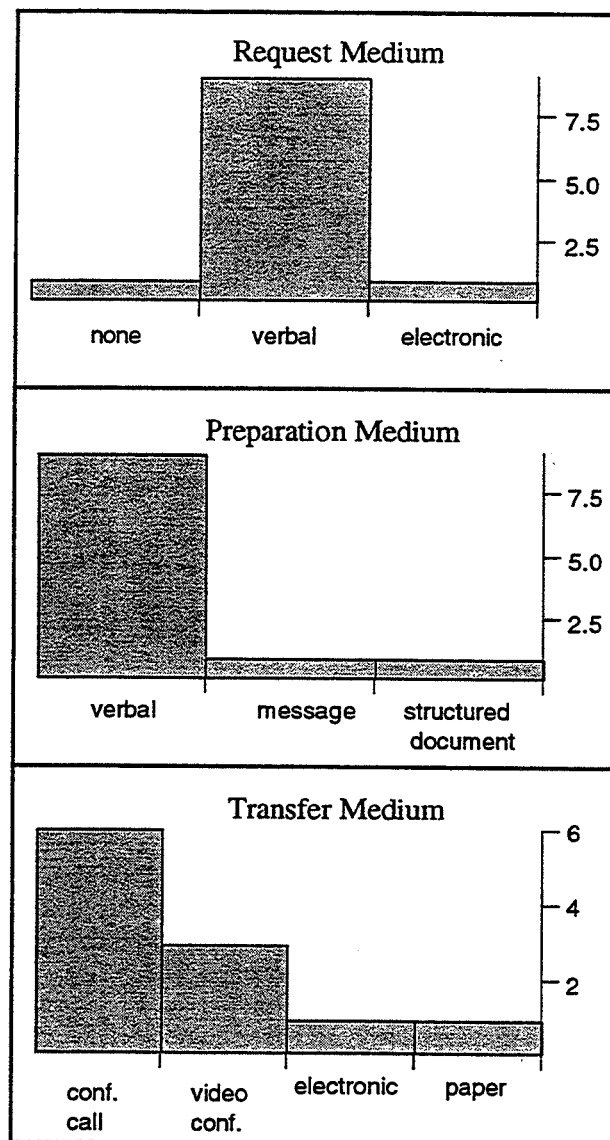


Figure 15. The different communication media used in the 11 highest-effort interactions.

Nearly all of the high-effort interactions involved a verbal request for information ($Mr=1$), no written preparation of the information ($Mp=2$), and were executed using a conference call or video conference ($Mt=2$ or 3). These patterns in the use of communication media, shown in Figure 15, differ dramatically from the patterns seen in the data as a whole. Interactions which involved a verbal request and no preparation usually took place during a face-to-face meeting in which many people were present, which implicitly increases the communication effort. In those meetings in which conference calling or videoconferencing was used, the technology actually slowed down the process. Significant amounts of time were spent waiting for remote participants to find the right page, to clarify issues for remote participants, etc. Also, the communication technology was unfamiliar to some participants.

All of the high-effort interactions involved technical information. This would imply that developers do not spend a large amount of time in administrative (non-technical) communication. In this study, 40% of all interactions were administrative in nature and none of them were highly effort-intensive. In fact, over all 10 reviews studied, 96% of the effort spent in communication involved technical information.

Comparisons between high-effort interactions and the whole set of interactions in terms of the other blocking variables yield few surprising results. The information in most of the high-effort interactions was unstructured, medium to large in size, and of average or higher complexity. The different uses of information in the high-effort interactions were not very different from that in the entire set of interactions, nor were the skill levels of the participants.

One other variable deserves a little more attention. The median number of participants in high-effort interactions is 10, but the median in the larger set of interactions is about half that (5.5). This result is not so straightforward as it might seem, however, because the variable N (number of participants) is not completely independent from Communication Effort. For some interactions, in fact, N is used in the calculation of CE . For example, CE for the interaction of type **discussion** is calculated by multiplying the amount of time spent in general discussion during the review meeting by N . To investigate whether or not the number of participants has an independent effect on effort, we normalized Communication Effort by dividing it by N . Then we picked the 15 interactions with the highest normalized CE (15 was the smallest number which included the 11 interactions we analyzed before as the highest-effort). The median number of participants in this subset is 8, lower than 10, but still considerably higher than the median of the data as a whole (5.5). So it appears that the highest-effort interactions involve more participants than interactions in general, regardless of which way effort is calculated.

In some of the discussion below, we refer both to "normalized" and "unnormalized" values of Communication Effort (CE). CE values are normalized simply by dividing them by N , as in the discussion above.

4.4 Interaction Types

Many of the types of interactions (**defects**, **review_material**, etc.) in this study differ in character from each other. Some of our most interesting results have come from studying each interaction type in isolation. Table 7 shows all the interaction types and relevant statistics, sorted by mean Communication Effort (unnormalized). Statistics based on normalized (by number of participants) CE are also shown for each interaction type.

Interaction Type	N	Unnormalized				Normalized			
		Mean	S.D.	Min	Max	Mean	S.D.	Min	Max
defects	11	642	658	92	1919	67	68	18	240
questions	11	409	373	68	1109	43	37	17	139
review_material	11	313	164	120	729	35	26	16	104
discussion	9	195	241	12	694	18	20	2	53
schedule_meeting	9	105	188	20	600	14	24	1	75
comments	7	97	92	9	273	41	46	3	137
choose_participants	2	65	78	10	120	33	39	5	60
rework	2	30	0	30	30	15	0	15	15
commented_material	7	22	22	4	60	9	8.8	1	23
preparation_done	3	20	0	20	20	4	1	2	5
summary_form	10	13	10	3	32	6	5	2	16
sum_form_rework	9	11	2	10	15	6	1	5	8
re-review_decision	9	8	4	4	13	1	0	0	1

Table 7. Mean Communication Effort by type of interaction, both unnormalized and normalized by the number of participants

The **defects** interaction is, on average, the most effort-intensive interaction type, whether or not Communication Effort is normalized by the number of participants. This makes intuitive sense, since this interaction embodies the entire purpose of the review. In our data set, most of the **defects** interactions involved a set of participants that fell into the second category (low **MOD**/high **XOD**), including all of those with **CE** above the mean. All of the **defects** interactions were verbal, and took place during a face-to-face meeting, conference call, or videoconference. The highest-effort **defects** interactions took place during conference calls. **Defects** interactions included anywhere from 4 to 15 participants, with a mean of about 9 participants. All of the **defects** interactions with (normalized or unnormalized) **CE** above the mean had 7 or more participants.

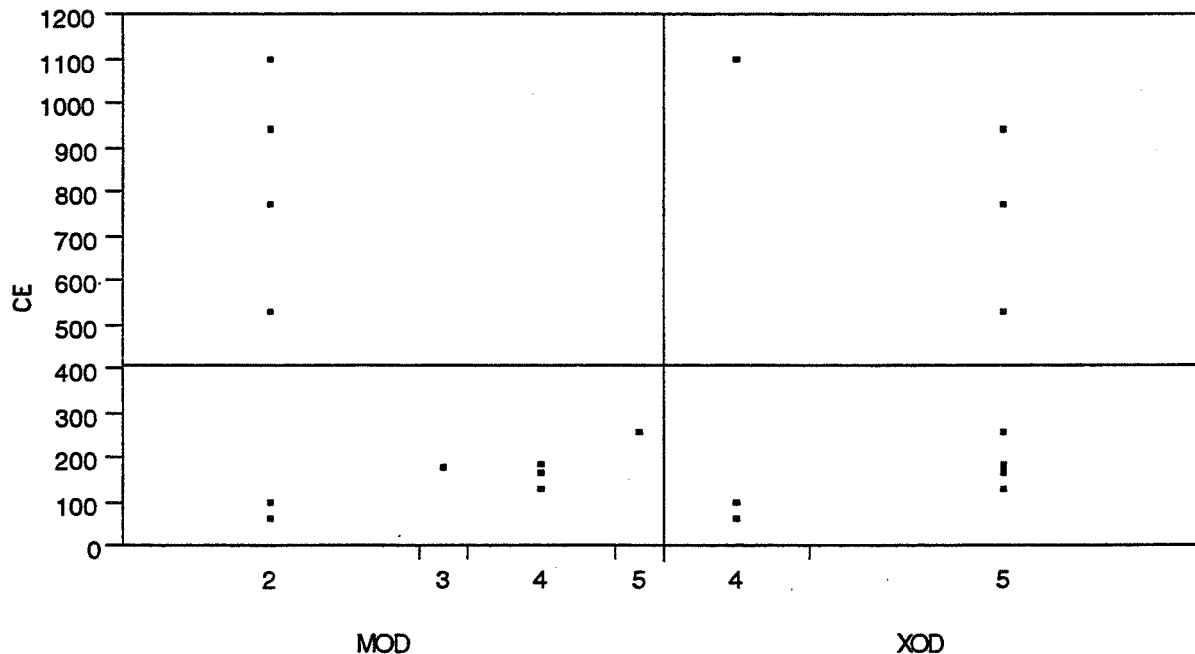


Figure 16. Communication Effort plotted against median and maximum Organizational Distance for questions interactions only

Another effort-intensive interaction type is the **questions** interaction. Again, the highest-effort interactions of this type fall into the second category of participant sets (low MOD/high XOD). This can be seen clearly in Figure 16.

Although all of the **questions** interactions have fairly high Physical Distance (3 or higher), the highest-effort ones have the highest Physical Distance, 4. Like the **defects** interactions, the above-average effort intensive **questions** interactions all had 7 or more participants.

The **discussion** interactions tend to be less effort-intensive than the **questions** or **defects** interactions, but still require more effort than most interaction types. **Discussion** interactions exhibit the same patterns in Organizational and Physical Distance as mentioned above for the **questions** and **defects** interactions. In addition, high-effort **discussion** interactions tend to involve information of relatively high complexity (**Comp**=4 or higher) and large size (**Size**>=4).

The **defects**, **questions**, and **discussion** interactions constitute all of the technical communication that takes place during a review meeting. The effort recorded for these interactions includes the effort required to prepare for, carry out, and digest this technical information. Since these interactions form the core of the work of a review, it is comforting to know that they are the ones which require the most effort. In fact, over all 10 reviews studied, 70% of the total Communication Effort expended was expended in interactions of these three types.

Two other relatively high-effort types of interactions, **review_material** and **comments**, exhibit slightly different behavior than described above. First of all, most of the high-effort **review_material** interactions have participants which fall into the third category described earlier (organizationally distant). The sets of participants for the high-effort **comments** interactions, on the other hand, fall into the first category (organizationally close). This contrasts with the observation that the participants in high-effort **defects**, **questions**, and **discussion** interactions are all in the second category. Another difference is that there is no apparent relationship between Communication Effort and Physical Distance for these types of interactions.

These results must be interpreted remembering that the set of participants in the **defects**, **questions**, and **discussion** interactions for each review is different than the set of participants for the **review_material** and **comments** interactions. The first three interactions take place during the review meeting, and the participants comprise all those present at the meeting. Furthermore, all the distance measures are calculated using every pair of participants. The distance measures for **review_material** interactions, on the other hand, reflect only the distances between the Author(s) and each Reviewer (i.e. not between Authors or between Reviewers). Thus, saying that the participants in a **review_material** interaction are organizationally distant means that the Authors are organizationally distant from the Reviewers. Similarly, the participants in a **comments** interaction are the Moderator and the Author(s). So saying that a **comments** interaction has a low Physical or Organizational Distance refers only to the distance between the Moderator and the Author(s).

The **review_material** and **comments** interactions also exhibited some relationships between effort and some of the blocking variables which did not seem relevant with other types of interactions. For instance, the high-effort **review_material** interactions all involved material that was highly structured and large, and took place during reviews in

which the Chief Reviewer was highly skilled. High-effort **comments** interactions all involved information that was more complex than average.

5. Limitations of the study

The major limitation of this study is its size and scope. It examines only 10 reviews, during one month, in a single development project. The amount of data collected (100 data points), relative to the number of relevant variables, is too small to assess statistical significance of many of the findings or to generalize the results in any way.

The three-part model built at the beginning of the study (see section 3.3) was extremely useful throughout as a framework for organizing the data and for communicating with developers. However, it could have been more useful. In particular, handling the data associated with interactions was cumbersome and limited somewhat the analyses which could be done easily. Some automatic support for managing this part of the model (or even handling the data itself through the model), as well as a better notation, was needed.

Another lesson learned from this study was that the interactions, as defined, did not naturally fit the way the participants thought about the review process. This made collecting and validating the data very difficult. For example, the Reviewers' preparation time had to be divided over several different interactions in order to fit the model. Some of it was included in the Communication Effort for the **defects** interaction, some for the **questions** interaction, etc. During the interviews, we asked some Reviewers how they divided their preparation time. We used their responses as a guideline, but we cannot be sure that the percentages are accurate or consistent. Modeling more in accordance with the process as it is enacted, and at a slightly higher level of abstraction, would help eliminate doubts about the accuracy of the data.

The design of the research variables and their levels in the pilot study was based on expert opinion and the literature, but the process of designing these measures was not very formal or well-documented. A more rigorous qualitative analysis is needed to support the design choices. Such an "ahead-of-time" analysis is part of what is called *prior ethnography*, a technique from qualitative research methods.

During data collection, the follow-up interviews after the observed reviews were vitally important. However, they could have been combined into just one interview for each interviewee. Instead, the questions were spread over several interviews over a period of 10 months. This led to memory, personnel turnover, and discontinuity problems. A single interview, as shortly after the review as possible, is preferred.

The observations in this study were not as rigorous as they could have been. The single observer was not very familiar with the application domain, and this sometimes made it difficult to determine what type of discussions were taking place during observations. As well, no reliability techniques were employed, such as audio- or videotaping the reviews, or having a second observer. This would have ensured better accuracy of the data. Also related to data accuracy, there were some variables that had no triangulation [Lincoln85] source. That is, there was only one data source for these variables. It would be better, and should be possible, to have at least two sources for each piece of information collected.

During observations and interviews, some field notes were taken in addition to the information on the interview forms and observation checklists. However, this data was not extensive or reliable enough to be used as part of the data analysis. If more faithful notes had been kept, this qualitative data could have been used to help explain and interpret the

quantitative results. The collection of useful anecdotes and quotes would also have been facilitated by making the interview questions more open-ended, that is, by relaxing the structuredness of the interviews a little.

One of the goals of this study was to serve as a pilot for a larger study begun recently. Although small, this pilot study was valuable in clarifying a number of issues related to how this subject is best studied. The limitations discussed above have been remedied in the design of the larger study, which is being conducted at NASA Goddard Space Flight Center. The main differences between this study and the pilot study described in this paper are its setting, size, and scope. Other differences arise as a result of remedying the limitations described above. The main goal of this larger study, as in the pilot study, is to learn how organizational structure characteristics affect the amount of effort expended on communication.

The setting for the larger study is the team developing AMPT, a mission planning tool. This project involves about 15-20 developers, and is the subject project for another experiment exploring a new software development process, Joint Application Development. As in the pilot study, we are studying the review process (called the inspection process at NASA) in particular. However, we expect to observe a much larger number of reviews (on the order of 30-50) over a longer period of time (3-6 months beginning December 1995).

6. Discussion and Summary

We have addressed the broad problem of organizational issues in software development by studying the amount of effort developers expend in certain types of communication. We have described an empirical study conducted to investigate the organizational factors that affect this effort. The research design combined quantitative and qualitative methods in an effort to be sensitive to uncertainty, but also to provide well-founded results. These methods include participant observation, interviewing, coding, graphical data displays, and simple statistical tests of significance.

Our findings are best summarized as a set of proposed hypotheses. The results of this study point to the validity of these hypotheses, but they are yet to be formally tested. Many of the methods and measures described in this paper may be used to do so. However, even as untested hypotheses, these findings provide important preliminary insight into issues of organizational structure and communication in software development:

- H1** Interactions tend to require more effort when the participants are not previously familiar with each others' work. This is consistent with Krasner's [Krasner87] findings about "common internal representations".
- H2** Interactions tend to require more effort when the participants work in physically distant locations. Curtis [Curtis88] and Allen [Allen85] have had similar findings.
- H3** Interactions which take place during a meeting (a verbal request and an unprepared reply) tend to require more effort than other interactions.
- H4** Interactions which involve some form of communication technology (conference calling and videoconferencing in this study) tend to require more effort.
- H5** Non-technical (administrative) communication accounts for very little (less than 5%) of the overall communication effort.
- H6** More participants tend to make interactions more effort-intensive, even when the effort is normalized by the number of participants.
- H7** In interactions that take place in a meeting, more effort is required when the set of participants includes mostly organizationally close members, but with a few

organizationally distant members. This contrasts with Curtis [Curtis88], who hypothesized that the relationship between organizational distance and communication ease is more straightforward.

- H8 Preparing, distributing, and reading material tends to take more effort when all participants are organizationally distant, and when the material is highly structured and large.
- H9 Writing and distributing comments to authors during the review meeting tends to take more effort when all of the participants are organizationally close, and when the material being reviewed is complex.

Recall that the problem to be addressed is the lack of knowledge about how to manage information flow in a software development organization and process. This study is not sufficient to solve this problem, but it is a first step. In section 1.1, several symptoms of this problem were described. The first is the difficulty of planning for communication costs. The findings of this study could be used to help in planning by pointing out characteristics which increase communication costs in reviews. For example, more than average time should be allowed for review meetings in which most of the participants are organizationally close, but a few are from distant parts of the organization. Alternatively, assignments could be made in such a way as to avoid such configurations of review participants.

The second symptom of the process information flow problem is that we do not know how to identify or solve communication problems as they arise. For example, if during the course of a project, developers are spending much more time preparing for reviews than planned, the findings above indicate that the problem may be that the participants are too organizationally distant, or that the material is too large. The problem might be solved by choosing reviewers who are closer, or by breaking the material to be reviewed into smaller pieces.

The third point raised in section 1.1 as a consequence of the research problem is that of learning from experience. This study represents a very small first step in building the experience necessary to effectively manage information flow in software development organizations. The next step for the authors is the larger empirical study described briefly in section 5. But there are several next logical steps in this line of research. No attempt has been made in this study to determine how communication effort affects software quality or development productivity. An understanding of this issue is necessary for effective management support. As well, this study does not address the issue of communication *quality*, only *quantity*. One cannot assume that the two are equivalent. Finally, there needs to be more work in the area of actually applying this new knowledge to the improvement of software development projects, and the mechanisms needed to achieve such improvement.

References

- [Allen85] Thomas J. Allen. *Managing the Flow of Technology*. The MIT Press, 1985.
- [Ballman94] Karla Ballman and Lawrence G. Votta. "Organizational Congestion in Large-Scale Software Development". In *Proceedings of the 3rd International Conference on Software Process*, pages 123-134, Reston, VA, October 1994.
- [Barley90] Stephen R. Barley. "The Alignment of Technology and Structure through Roles and Networks". *Administrative Science Quarterly*, 35:61-103, 1990.
- [Bradac94] Mark G. Bradac, Dewayne E. Perry, and Lawrence G. Votta. "Prototyping a Process Monitoring Experiment". *IEEE Transactions on Software Engineering*, 20(10):774-784, October 1994.
- [Curtis88] Bill Curtis, Herb Krasner, and Neil Iscoe. "A Field Study of the Software Design Process for Large Systems". *Communications of the ACM*, 31(11), November 1988.
- [Davenport90] Thomas H. Davenport and James E. Short. "The New Industrial Engineering: Information Technology and Business Process Redesign". *Sloan Management Review*, pages 11-27, Summer 1990.
- [Ebadi84] Yar M. Ebadi and James M. Utterback. "The Effects of Communication on Technological Innovation". *Management Science*, 30(5):572-585, May 1984.
- [Galbraith77] Jay R. Galbraith. *Organization Design*. Addison-Wesley, 1977.
- [Gilgun92] Jane F. Gilgun. "Definitions, Methodologies, and Methods in Qualitative Family Research". In *Qualitative Methods in Family Research*. Sage, 1992.
- [Hammer90] Michael Hammer. "Reengineering Work: Don't Automate, Obliterate". *Harvard Business Review*, pages 104-112, July 1990.
- [Krasner87] Herb Krasner, Bill Curtis, and Neil Iscoe. "Communication Breakdowns and Boundary Spanning Activities on Large Programming Projects". In Gary Olsen, Sylvia Sheppard, and Elliot Soloway, editors, *Empirical Studies of Programmers*, second workshop, chapter 4, pages 47-64. Ablex Publishing, New Jersey, 1987.
- [Liker86] Jeffrey K. Liker and Walton M. Hancock. "Organizational Systems Barriers to Engineering Effectiveness". *IEEE Transactions on Engineering Management*, 33(2):82-91, May 1986.

- [Lincoln85] Yvonna S. Lincoln and Egon G. Guba. *Naturalistic Inquiry*. Sage, 1985.
- [March58] J.G. March and Herbert A. Simon. *Organizations*. John Wiley, New York, 1958.
- [Mintzberg79] Henry Mintzberg. *The Structuring of Organizations*. Prentice-Hall, 1979.
- [Perry94] Dewayne E. Perry, Nancy A. Staudenmayer, and Lawrence G. Votta. "People, Organizations, and Process Improvement". IEEE Software, July 1994.
- [Rank92] Mark R. Rank. "The Blending of Qualitative and Quantitative Methods in Understanding Childbearing Among Welfare Recipients". In *Qualitative Methods in Family Research*, Sage, 1992.
- [Sandelowski92] Margarete Sandelowski, Diane Holditch-Davis and Betty Glenn Harris. "Using Qualitative and Quantitative Methods: The Transition to Parenthood of Infertile Couples". In *Qualitative Methods in Family Research*, Sage, 1992.
- [Schilit82] Warren Keith Schilit. "A Study of Upward Influence in Functional Strategic Decisions". PhD thesis, University of Maryland at College Park, 1982.
- [Schneider85] Larissa A. Schneider. "Organizational Structure, Environmental Niches, and Public Relations: The Hage-Hull Typology of Organizations as a Predictor of Communication Behavior". PhD thesis, University of Maryland at College Park, 1985.
- [Stinchcombe90] Arthur L. Stinchcombe. *Information and Organizations*. University of California Press, 1990.
- [Sullivan85] Matthew J. Sullivan. "Interaction in Multiple Family Group Therapy: A Process Study of Aftercare Treatment for Runaways". PhD thesis, University of Maryland at College Park, 1985.
- [Taylor84] Steven J. Taylor and Robert Bogdan. *Introduction to Qualitative Research Methods*. John Wiley and Sons, 1984.

Understanding and Predicting the Process of Software Maintenance Releases

Victor Basili, Lionel Briand, Steven Condon,
Yong-Mi Kim, Walcélio L. Melo and Jon D. Valett†

53-61

415 769

Abstract

One of the major concerns of any maintenance organization is to understand and estimate the cost of maintenance releases of software systems. Planning the next release so as to maximize the increase in functionality and the improvement in quality are vital to successful maintenance management. The objective of this paper is to present the results of a case study in which an incremental approach was used to better understand the effort distribution of releases and build a predictive effort model for software maintenance releases. This study was conducted in the Flight Dynamics Division (FDD) of NASA Goddard Space Flight Center (GSFC). This paper presents three main results: 1) a predictive effort model developed for the FDD's software maintenance release process, 2) measurement-based lessons learned about the maintenance process in the FDD, 3) a set of lessons learned about the establishment of a measurement-based software maintenance improvement program. In addition, this study provides insights and guidelines for obtaining similar results in other maintenance organizations.

Keywords: *software maintenance, measurement, experience factory, case studies, quality improvement and goal/question/metric paradigms.*

1. Introduction

1.1 Issues

360832

111

Software maintenance is generally recognized to consume the majority of resources in many software organizations [Abran&Nguyenkim 1991; Harrison&Cook 1990]. As a result, planning releases so as to maximize functionality and quality within the boundaries of resource constraints (such as, budget, personnel, and time to market) is vital to the success of an organization. The software maintenance process is, however, still poorly understood and loosely managed worldwide. As described in [Haziza *et al.* 1992], numerous factors can affect software maintenance quality and productivity, such as process, organization, experience, and training. Unfortunately the complexity of the phenomena frequently obscures the identity and impact of such factors in any given maintenance organization. The resulting uncertainty about productivity and quality in the next software release gives rise to unreliable cost and schedule release estimates.

To effectively manage the software release process, managers must be supplied with more accurate information and more useful guidelines to aid them in improving the decision-making process, planning and scheduling maintenance activities, foreseeing bottlenecks, allocating resources, optimizing the implementation of change requests by releases, etc. In order to accomplish this, we need to define and validate methodologies that take into account the specific characteristics of a software maintenance organization and its processes, e.g., the software maintenance release process. However, methods that help software maintainers change large software systems on schedule and within budget are scarce. Methods currently available for improving software processes, such as the Software Engineering Institute Capability Maturity Model (SEI CMM) [Paulk *et al.* 1993], have not been validated thoroughly. Even though a few methods have been demonstrated to be useful for software development (e.g., QIP [Basili&Rombach 1988]) they have only recently begun to be applied to software maintenance [Valett *et al.* 1994]. The work described in this paper is a further step in the application of these methods.

† Authors are listed in alphabetical order. V. Basili, Y.-M. Kim and W. Melo are with the University of Maryland, Institute for Advanced Computer Studies and Computer Science Dept., A. V. Williams Bldg., College Park, MD 20742, USA. S. Condon is with Computer Sciences Corporation, 10110 Aerospace Rd., Lanham-Seabrook, MD 20706, USA. L. Briand is with CRIM, Montreal, Canada. J. Valett is with NASA Goddard Space Flight Center, Software Engineering Branch, Greenbelt, MD 20771, USA. E-mail: {basilikimymelo}@cs.umd.edu, lbriand@crim.ca, steven_condon@cscmail.csc.com, jon.valett@gsfc.nasa.gov

Copyright 1996 IEEE. Published in the Proceedings of the 18th International Conference on Software Engineering (ICSE-18), March 25-29, 1996, Berlin, Germany. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works, must be obtained from the IEEE.

1.2 Objective

The objective of this paper is to use an incremental and inductive approach for improving software maintenance by focusing on the construction of descriptive and predictive models for software maintenance releases. We present the results of a case study in which this approach was successfully used to build a predictive effort model for software maintenance releases in a large-scale software maintenance organization. This case study took place in the Flight Dynamics Division (FDD) of the NASA Goddard Space Flight Center (GSFC). This organization is a representative sample of many other software maintenance organizations. The FDD maintains over one hundred software systems totaling about 4.5 million lines of code, and many of these systems are maintained for many years and regularly produce new releases.

In this paper, we are mostly concerned with presenting the results of the process used to build descriptive and predictive models of software maintenance releases in a particular environment. Although the models produced in this study are organization-dependent, we believe that the process used to build them can be easily replicated in different software organizations.

The paper is organized as follows. It first presents the framework in which this study was conducted: the FDD and the Software Engineering Laboratory (SEL). Next an overview of our approach to software maintenance process improvement is provided. The paper then presents the measurement program used to collect product and process data about maintenance projects and releases. This is followed by a quantitative analysis of the data collected from January 1994 to June 1995 on the delivery process of over 29 releases of 11 different systems. This analysis presents descriptive models of the maintenance environment, as well as a predictive model for release productivity. Next the paper presents the lessons learned from the analysis and validation of data, and discusses lessons drawn from establishing a software maintenance measurement program. Finally, future work is outlined.

2. Study framework and approach to model building

2.1 The environment

GSFC manages and controls NASA's Earth-orbiting scientific satellites and also supports Space Shuttle flights. For fulfilling both these complex missions, the FDD developed and now maintains over 100 different software systems, ranging in size from 10 thousand source lines of code (KSLOC) to 250 KSLOC, and totaling approximately 4.5 million SLOC. Many of these systems are maintained over

many years and regularly produce new releases. Of these systems, 85% are written in FORTRAN, 10% in Ada, and 5% in other languages. Most of the systems run on IBM mainframe computers, but 10% run on PCs or UNIX workstations.

This study was conducted through the SEL, which is a joint-venture between GSFC, Computer Sciences Corporation, and the University of Maryland. Since 1976, the SEL has been modeling and experimenting in the FDD with the goals of understanding the software development process in this environment; measuring the effect of software engineering methodologies, tools, and models on this process; and identifying and applying successful practices [McGarry *et al.* 1994]. Recently, responding to an organizational need to better control the cost and quality of software maintenance, the SEL has initiated a program aimed at characterizing, evaluating and improving these maintenance processes.

2.2 The approach

This SEL program on maintenance began in October 1993 and is being conducted using an empirical approach which is an instantiation of the more general Quality Improvement Paradigm (QIP) and the Goal/Question/Metric Paradigm (GQM) [Basili&Rombach 1988]. In the following paragraphs we provide an overview of this approach and show how it has helped us in the construction of a predictive model for software maintenance releases. This approach was tested and continuously refined through experience. Further details can be found in [Briand *et al.* 1994; 1995].

First, qualitative studies were performed in order to better comprehend organization- and process-related issues. Here, the objective was to identify and understand, as objectively as possible, the real issues faced by the organization. Specific modeling techniques such as the Agent Dependency Model were used as part of this step (see [Briand *et al.* 1995]). Such a technique can help capture important properties of the organizational context of the maintenance process and help to understand the cause-effect mechanisms leading to problems. Such qualitative data must be complemented with quantitative data.

In a subsequent step, the outputs produced by the first step were used to justify and define a relevant and efficient measurement program (i.e., what to collect, when to collect, and how to collect). In addition, interpreting the data coming from such a program was made easier because of the increased level of understanding of the process in place.

Once the measurement program began (i.e., data collection forms were available, data collection procedures defined, people trained, etc.), process and product data were collected and various issues identified as relevant to the maintenance process were analyzed. Based upon such analyses, the relationships between process attributes, such as effort, and other variables

characterizing the changes, the product to be changed, and the change process were identified. For instance, in this paper, a model for predicting release effort from estimated release size is presented to help software maintenance managers in the FDD environment optimize release resource expenditures. Such models will be incrementally refined when new information of either a qualitative or quantitative nature is available.

3. A QQM for this study

As pointed out in [Pigoski&Nelson 1994; Rombach *et al.* 1992; Schneidewind 1994], the establishment of a measurement program integrated into the maintenance process, when well defined and established, can help us acquire an in-depth understanding of specific maintenance issues and thereby lay a solid foundation for the improvement of the software maintenance release processes. To do so, we must define and collect those measures that would most meaningfully characterize the maintenance process and products. In order to define the metrics to be collected during the study, we used the QQM paradigm [Basili&Rombach 1988]. We first present the QQM goals of the study, then present the metrics and the data collection method used. For the sake of brevity the questions accompanying each goal are presented with the data analysis.

3.1 Goals

Goal 1: Analyze: the maintenance release generation process
for the purpose of: characterization
with respect to: effort
from the point of view of: management,
experience factory for maintenance

In this goal, we are interested in understanding the maintenance release generation process of the maintenance organization with respect to the distribution of effort across software activities, across maintenance change types, and across software projects. Next, we need to identify the variables we can use to produce predictive models for maintenance. That is, we must study and understand the relationship between the different facets of effort and other metrics, such as type of releases, type of change, change size, types of component change (modification, inclusion or deletion of code). Formalizing such a problem in the QQM format, we formulate the following goal:

Goal 2: Analyze: maintenance release process
for the purpose of: identifying relationships
between effort and other variables
with respect to: type of release, type of
change, size of change, and kind of change

from the point of view of: experience factory
for maintenance

This second goal is a necessary step that leads from Goal 1 to the following goal:

Goal 3: Analyze: release delivery process
for the purpose of: prediction
with respect to: productivity
from the point of view of: experience factory
for maintenance

3.2 Metrics and models

In this section we describe the metrics and models used in this study. The preliminary qualitative modeling of the maintenance process enabled the definition and refinement of these metrics and models.

Maintenance change types

We consider the following maintenance change types:

- error correction: correct faults in delivered system.
- enhancement: improve performance or other system attributes, or add new functionality.
- adaptation: adapt system to a new environment, such as a new operating system.

Maintenance activities

The following maintenance activity classification is used in the data collection forms:

- Impact analysis/cost benefit analysis. The number of hours spent analyzing several alternative implementations and/or comparing their impact on schedule, cost, and ease of operation.
- Isolation. The number of hours spent understanding the failure or request for enhancement or adaptation.
- Change design. The number of hours spent actually redesigning the system based on an understanding of the necessary change; includes semiformal documentation, such as release design review documents.
- Code/unit test. The number of hours spent to code the necessary change and test the unit; includes semiformal documentation, such as software modification test plan.
- Inspection/certification/consulting. The number of hours spent inspecting, certifying, and consulting on another's design, code, etc., including inspection meetings.
- Integration test. The number of hours spent testing the integration of the components.
- Acceptance test. The number of hours spent acceptance testing the modified system.

- Regression test. The number of hours spent regression testing the modified system.
- System documentation. The number of hours spent writing or revising the system description document and math specification.
- User/other documentation. The number of hours spent writing or revising the user's guide and other formal documentation, except system documentation.
- Other. The number of hours spent on activities other than the ones above, including management.

A more detailed presentation of the maintenance activities model is presented in [Valett *et al.*, 1994].

Release types

Maintenance releases in our environment were classified into three categories: mostly error correction, mostly enhancement, and mixture. A more detailed discussion is presented in Section 4.4

Size and effort

The size of a software change is measured as the sum of the number of source lines of code (SLOC) added, changed, and deleted. SLOC is defined to include all code, unit header lines, comments, and blank lines. Effort is measured by person hours that were charged to maintenance projects.

3.3 Data collection method

The following forms were used to collect the data for this study:

- software change request (SCR) form;
- weekly maintenance effort form (WMEF);
- software release estimate form (SREF).

Again, without a preliminary qualitative analysis of the maintenance process, determining the content and format of the WMEF and SREF forms would have been extremely difficult.

3.3.1 SCR forms

On the SCR, the user or tester specifies what *type of change* is being requested: *error correction*, *enhancement*, or *adaptation*. The maintainer specifies using an ordinal scale the effort spent isolating/determining the change, as well as the effort spent designing/implementing/testing the change. The maintainer also provides six numbers characterizing the extent of the change made: (1) number of SLOC added, (2) changed, (3) deleted; (4) number of components added, (5) changed, (6) deleted. In addition, the maintainer further specifies how many of the components in item (4) were

newly written, how many were borrowed and reused verbatim, and how many were borrowed and reused with modification

3.3.2 WMEF forms

Each maintainer, tester, and manager working on one of the study projects was required to report project hours each week on a WMEF. The WMEF required each person to break down project effort two ways: (1) by specifying the hours by the type of change request performed (error corrections, enhancements, or adaptations) or as *other* hours (e.g., management, meetings), and (2) by specifying the hours by the software activities performed (such as design, implementation, acceptance testing).

Because the WMEF did not originally allow a person to specify to which maintenance release of the project his hours applied, uncertainty resulted if a maintenance team was involved in more than one maintenance release in the same week. For many projects, maintenance releases did overlap. Therefore, in August 1994, we revised the WMEF by requiring personnel in the study to specify to which release each activity hour applied. In addition, each maintainer (but not tester) is now required to specify on his WMEF to which SCR each activity hour applies.

3.3.3 SREF forms

The SREF is a new form created by the authors to capture estimates of the release schedule, release effort, release content (i.e., list of SCRs), and release extent (i.e., number of units and lines of code to be added, changed or deleted). Maintenance task leaders submit an SREF at the end of each phase in the maintenance release life cycle

4. Quantitative analysis of the SEL sample maintenance goals

In this section, we provide the results of our analyses from the data collected during this study. In most cases, the data consisted of 25 complete releases for ten different projects. The effort per release ranged from 23 hours to 6701 hours, with a mean of 2201 hours. The total changes per release ranged from 21 SLOC to 23,816 SLOC, with a mean of 5654 SLOC.

4.1 Effort across maintenance activities

In this section we are interested in the following questions related to Goal 1:

- Q1.1. What is the distribution of effort across maintenance activities (i.e., analysis/isolation, design, implementation, testing, and other; see below)?

- Q1.2. What are the costliest projects and what is the distribution of effort across maintenance activities in these projects?

For simplicity, we have grouped the 12 maintenance activity categories into 5 groups, as follows:

- Analysis/isolation: impact analysis/cost benefit analysis, isolation
- Design: change design, 1/2 (inspection/certification/consulting)
- Implementation: code/unit test, 1/2 (inspection/certification/consulting)
- Testing: integration test, regression test, acceptance test
- Other: system documentation, other documentation, other

Using these groupings, the distribution of maintenance effort across maintenance activities is shown in Figure 1. The first pie chart of this figure represents the overall distribution based on the total effort expended in the 25 complete releases (10 projects) studied. Five projects accounted for 17 of these 25 releases. The remaining pie charts show the effort distributions for these 5 projects, based on their 17 complete releases. These 5 projects were the costliest projects in the FDD between January 1994 and June 1995, when counting all project effort, i.e., including effort for both complete and partial releases in this time period. During this time period, Swingby accounted for 28% of the maintenance effort, MTASS for 19%, GTDS for 12%, MSASS for 10%, and ADG for 8%.

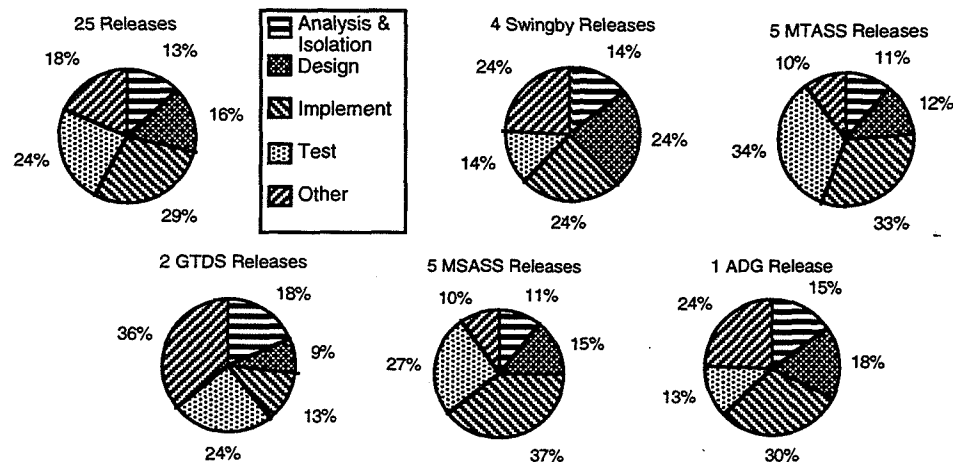


Figure 1: Distribution of effort among software maintenance activities

One difference among these projects is that MTASS has the largest percentage of testing effort, 34%. Closer examination reveals that testing made up 17% of the effort of the two earlier MTASS releases and 43% of the three latter releases. A similar trend is suggested by MSASS—17%, followed by 31%. In addition, both projects show a decreasing trend in implementation effort, 43% followed by 27% for MTASS, and 41% followed by 37% for MSASS. These trends are not evident, however, for Swingby, the only other project in our study that is represented by more than 2 releases. The increase in MTASS and MSASS testing may be due to the fact that these systems consist of large software libraries that are enhanced and reused from mission to mission. As the software grows, more regression test time is necessary. Another difference is seen in the large amount of 'other' time for GTDS. One of the GTDS releases involved porting the GTDS software from an IBM mainframe to a workstation. A significant amount of training time (listed as other) may have been necessary for the

maintainers. More study is required before we can confidently recommend such pie charts to release managers as guides for resource allocation. It is likely that such models will also need to factor in what type of changes (adaptation, correction, or enhancement) constitute the release.

4.2 Effort across maintenance change types

In this section we consider the following questions related to Goal 1:

- Q1.3. What is the distribution of effort across maintenance change types (i.e., adaptation, error correction, enhancement, other)? That is, how was the total maintenance effort expended?
- Q1.4. Is the distribution of effort across maintenance activities the same for the different software maintenance change types?

Figure 2 presents the average distribution of effort across maintenance change types. The distributions for individual projects vary significantly from each other and also from this average distribution. For example, effort spent on enhancements varied from 51% to 89% (with a mean of 61%) among the most dominant projects.

In the FDD, enhancements typically involve more SLOC than error corrections. The 25 complete releases contained 187 change requests from users. Of these, 84 were enhancement change requests, with a mean size of 1570 SLOC, whereas 94 were error correction change requests, with a mean size of only 61 SLOC. This data supports the intuitive notion that error corrections are relatively small isolated changes, while enhancements are larger changes to the functionality of the system.

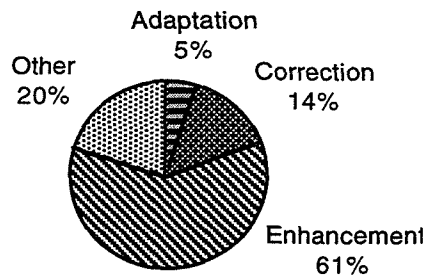


Figure 2: Effort Distribution by Type of Change

Now, we address the fourth question. In order to answer this question, we need to know how a maintainer's activity effort is distributed for each change type. With the old WMEF we could not simultaneously analyze effort by both activity and change type. With the new WMEF we can do so for the programmers' effort, because programmers report the activity effort associated with each SCR, and we know the change type of each SCR. Due to the fact that testers, and usually task leaders, report their effort by release—but not by SCR—we cannot analyze their effort this way.

broken down by maintenance activities. We do not include the 'testing' and 'other' groups of activities, because much of this activity is not tagged to individual SCRs, and we do not want to present a misleading picture of how much time is spent in these activities. As expected, software maintainers spent more effort on isolation activities when correcting code than when enhancing it. Conversely, they spent much more time on inspection, certification, and consulting, when enhancing code than when correcting it. The proportions of effort spent on design and code/unit test are almost the same for the two types of change requests.

Figure 3 shows the effort spent by programmers on correction and enhancement maintenance types, each

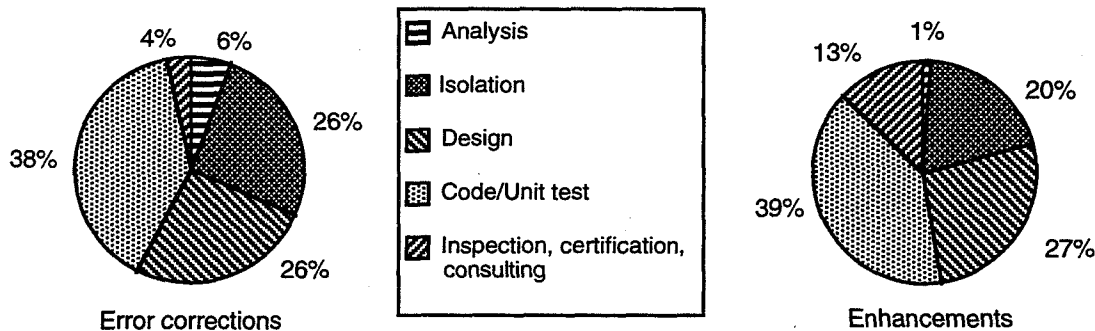


Figure 3. Programmer effort distribution across five maintenance activities for error correction and enhancement maintenance changes

4.3 Testing changes vs. release changes

In this section we consider the following question related to Goal 1:

- Q1.5. What is the impact of the errors inserted into the projects by the maintainers with respect to maintenance effort and code changed?

In this study we distinguished two types of change requests: user and tester change requests. The original content of the release consists of change requests

submitted by users. During the implementation of each release some errors may be introduced by the maintenance work. If these errors are caught by the testers, they in turn generate tester change requests, which become part of the same release delivery. The 25 complete releases contained 187 user change requests, which required 138,000 SLOC. The same releases had 101 tester change requests, which required 3600 SLOC. Thus the tester change requests accounted for 35% of the SCRs in the release, but only 2.5% of the SLOC, as is shown in Figure 4.

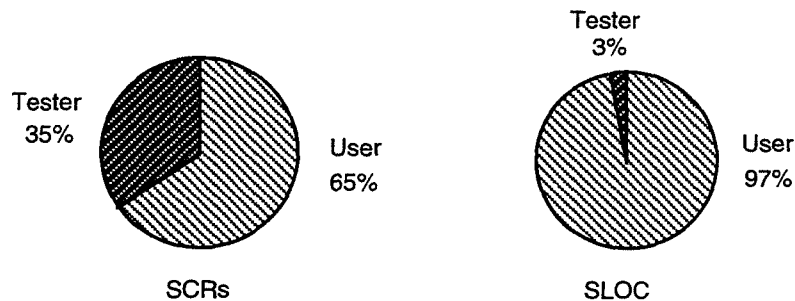


Figure 4: SCR count and SLOC differences between user and tester change requests (for 25 releases)

The effort data associated with individual SCRs is incomplete for releases which began before August 1994 (when the authors revised the WMEF), so the percent of effort associated with tester SCRs is unclear, but the SLOC count suggests that it is a small percentage. In a preliminary attempt to examine the distribution of effort between tester change requests and user change requests, the authors selected 5 releases started and completed between August 1994 and June 1995 (see Figure 5). Since enhancements tend to be larger than error corrections, and since all tester change

requests are error corrections, we ignored the enhancements requested by the users (there were no adaptations). In this sample 42% of the error correction SCRs are tester SCRs, but these tester SCRs account for only 27% of the programmer effort associated with the error correction SCRs in these 5 releases. The number of SLOC added, changed, or deleted for these tester SCRs corresponds to 29% of the total number of SLOC changed, added or deleted for all error correction SCRs.

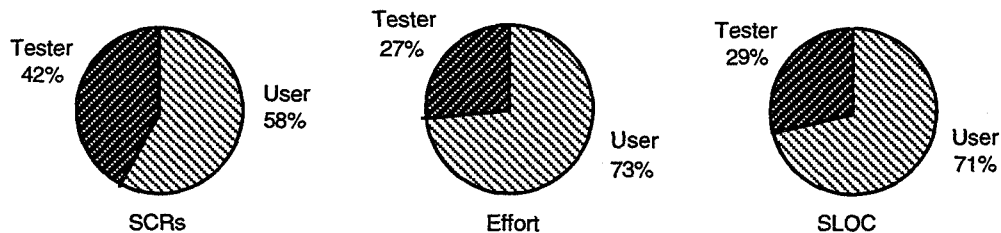


Figure 5: SCR count, Effort, and SLOC differences between 5 completed releases

In order to better comprehend the differences between user and tester SCRs with regard to effort and SLOC we calculated the level of significance of these differences. To do so, we used the Mann-Whitney U non-parametric tests [Hinkle *et al.* 1995].

We assumed significance at the 0.05 level, i.e., if the p value is greater than 0.05, then we assume there is no observable difference between tester and user SCRs. The results of these tests as well as other descriptive statistics are provided in Table 1. These statistics are shown for the sake of completeness and also because

they help us interpret the results of the analysis in the remainder of this section. In addition, these statistics will facilitate future comparisons of results in similar studies since they will help explain differences in results through differences in statistical distributions.

As this table shows, the mean productivity for user SCRs (3.50) is almost the same as for tester SCRs (3.76). Productivity is defined as the total SLOC added, changed and deleted, divided by the total effort spent to add, change, or delete that SLOC. Based on the results presented in Table 1, we can conclude that there is no significant difference between the user SCRs as compared to tester SCRs from the perspectives of effort, SLOC and productivity (all the p values are greater than 0.05). Therefore, even though the

maintainers already spent time understanding the code to be modified when the change was first requested, they are not significantly more productive when correcting their own mistakes than they were earlier correcting errors reported by the users. This surprising result is an additional motivation to eliminate errors introduced during the maintenance process. Understanding why tester SCRs are not easier to correct in the current maintenance process may lead to substantial productivity gains.

However, we cannot confirm if this is only a particular situation which happened on these 5 completed releases. We must continue to pursue this analysis in order to verify the validity of these results.

Descriptive Statistics	User SCRs			Tester SCRs			Mann-Whitney U Test	
	SLOC	hours	Produc.	SLOC	hours	Produc.	Variable	p-value
Maximum	300	68	27.27	75	23	16	SLOC	0.2069
Minimum	3	2	0.15	4	3	0.19	Hours	0.2637
Median	24	19	1.26	16	16	1.92	Productivity	0.3215
Mean	57	26.63	3.50	32.75	13.53	3.76		
Std Dev	89.45	22.82	7.94	31.05	7.72	5.21		

Table 1: Descriptive statistics of user and tester SCRs and Mann-Whitney U test results

4.4 Release productivity

For this paper, our major concern is how to estimate the cost of subsequent maintenance releases. Planning the next release so as to maximize the increase of functionality and the improvement of quality is vital to successful maintenance management. By analyzing the various relationships between effort and other variables (see Goal 2), we suggest for our environment a predictive model (Goal 3) based upon lines of code per release. By following our procedure (Goals 1, 2 and 3) other organizations can develop their own predictive models, based upon their specific characteristics and the relationships between variables found in their organization. In this section we are interested in answering the following questions related to Goal 3:

- Q3.1. What is the productivity model for the 3 different types of maintenance releases (i.e., enhancement, error correction, and mixture) within the SEL?
- Q3.2. Does a constant amount of overhead exist for any type of maintenance release?

Evaluating the data available on 25 completed maintenance releases within the SEL environment,

provided insight into potentially different kinds of maintenance releases. In attempting to develop a cost model for software maintenance releases, we first plotted the size of maintenance releases (measured in SLOC added, changed, and deleted) against the total effort expended on the release. Initial evaluation of this data (by visual inspection) showed that the data seemed to break into 4 different groups.

One group of 4 releases had very high productivity. In trying to find some reason to explain why these releases differed from the others, we noted that the average ratio of units *added* versus *changed* for these 4 releases (1.4) was much higher than for the other 21 releases (0.1). Were the added units primarily reused units (either verbatim or with modification), rather than newly written units, we might assume that the high productivity of these 4 releases was due to their high reuse. But the source of the units added to these 4 releases was not consistent. Sometimes the added units were predominantly borrowed from other projects and reused with modification. But other times the added units were predominantly newly written units. In the latter case, reuse is not the answer.

The answer may be the header, PDL*, comment, and blank lines which SLOC includes in its definition.

* In the FDD, pseudocode, referred to as Program Design Language (PDL), is included in the source code file.

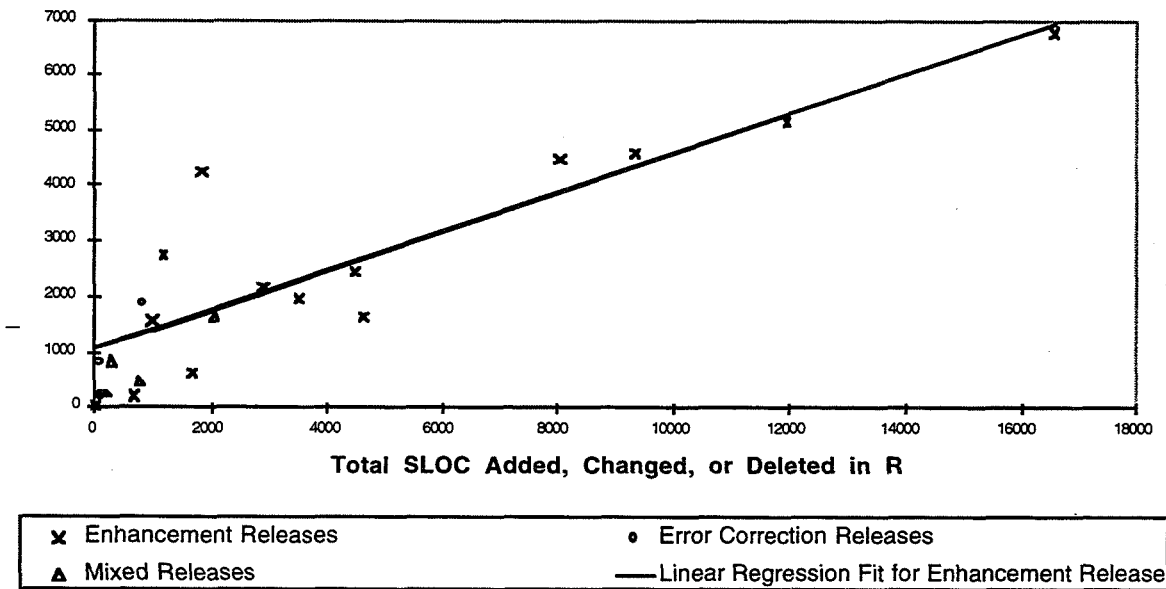


Figure 6. Linear Regression Results for Enhancement Releases

Newly written units typically contain a high percentage of such lines, but older units—and the maintenance changes made to them—often include a much smaller percentage of such lines. The older units often do not have PDL, so PDL is often not updated when the code is changed. Although more study is needed to verify this hypothesis, this reinforces the need to have a thorough knowledge of the process and products in order to interpret the data and build accurate models.

Dismissing these four releases as unusual, we continued to evaluate the remaining 21 releases. Based on an inspection of the data, we developed a scheme for characterizing the other 3 kinds of releases. The three groups seemed to be divided into those releases that were primarily made up of enhancements, those made up primarily of error corrections, and those that fell into neither of these two categories. The scheme used to divide the releases is based on the percentage of change requests within the release that were enhancements or corrections and the percentage of SLOC that was added, changed, or deleted as a result of enhancements or corrections. Two criteria are established for testing the release type:

- Criterion 1 - (Percentage of Change Requests that are enhancements > 80) or (percentage of SLOC due to enhancements > 80)
- Criterion 2 - (Percentage of Change Requests that are corrections > 80) or (percentage of SLOC due to corrections > 80)

Release type was then determined based on the following test:

```

If (criterion 1) and Not (criterion 2)
    then Release type = Enhancement
Elseif (Criterion 2) and Not (Criterion 1)
    then Release type = Correction
Else Release type = mixed
Endif

```

This test subdivided the remaining 21 releases into 14 enhancement releases, 3 correction releases, and 4 mixed releases.

The major result of this study is the development of a predictive cost model for maintenance releases that are primarily composed of enhancements. Figure 6 shows the results of a standard linear regression of total release effort versus total lines of code added, changed, and deleted. This model has a coefficient of determination (R^2) of 0.75, which is statistically significant at the 0.00006 level. By estimating the size of a release, an effort estimate can be determined. The equation for the line fit is:

$$\text{Effort in hours} = (0.36 * \text{SLOC}) + 1040$$

Any maintenance release will have some overhead. It is likely that this overhead stems partly from regression testing and comprehension activities which are somewhat independent from the size of the change. The y-intercept of 1040 hours seems to imply that there is an average release overhead of approximately 1040 hours for enhancement releases in the FDD.

The number of data points for error correction releases and mixed releases makes development of accurate models for them difficult. More data points

will be needed to determine if similarly accurate models can be developed. The preliminary data suggests, however, that the productivity for error correction releases and mixed releases is significantly lower than for enhancement releases. This suggests that error corrections are less productive—in terms of SLOC per hour—than are enhancements. The error correction releases and mixed releases tend to be smaller than most of the enhancement releases. The interpretation of these observations can result in different courses of action for the manager. If improving productivity is the main concern, then it may be wise to try to avoid scheduling small error correction releases. Instead the manager should try, when possible, to package small error corrections in a release with larger enhancements. If the enhancements require making changes to the same units or group of units as required by the error corrections, then the savings would likely be larger still. On the other hand, there may be criteria other than productivity to be considered: certain error corrections may be vital to a mission, and thus cannot be put off until another release, or the defect may be of such severity that unless the error correction is performed the system is unusable. The scheduling of error corrections will involve tradeoffs regarding productivity.

5. Limitations of the data collection and lessons learned

During this research effort, many valuable lessons were learned. These lessons can be divided into general results for studying maintenance and results for data collection.

In the area of lessons learned for studying maintenance, the following statements can be made:

- An overall understanding of the maintenance process and the maintenance environment is crucial to any maintenance study. The combination of qualitative understanding with quantitative understanding has been invaluable. The qualitative understanding helped to drive and improve the data collection process.
- Understanding the environment provides valuable context for the data analysis. Without a thorough understanding of the environment four outlier enhancement releases might not have been recognized as a distinct subset.

In the area of lessons learned on data collection:

- Recognize the limitations of the data and work within those limitations. Data collection by its nature is inexact. Researchers must work within the limits of the data and recognize that the conclusions are only as valid as the data. Qualitative evidence (i.e., structured interviews, analysis of products and process documentation)

should be actively used to gain more confidence in the results.

- Assuring the quality of the data collected is a difficult task.

The following paragraphs describe the quality assurance procedures and analysis of the quality of the data for this study. The SCRs are tracked very effectively by the FDD configuration management (CM) team. Their logging and tracking database provided a thorough check on release contents. By comparing the contents of the SEL database with the CM database, we were able to identify any SCRs missing from the SEL database. Copies of these missing SCRs were then acquired from the CM team and entered into the SEL database. Thus, in general, the release contents were characterized to a high level of confidence.

Two minor problems were encountered with the SCR data. First, from talks with maintainers it was learned that the maintainer does not always agree with the change type specified by the user or tester. The user may call a change request an error correction, whereas the maintainer might judge it to be an enhancement. This is not thought to occur in many cases.

Secondly, during the course of the study we learned that not all maintainers were using the same definition in reporting lines of code added, changed, or deleted. The SEL usually uses source lines of code (SLOC), which includes all PDL lines, comment lines, and blank lines, as well as regular lines of code. Most maintainers were using this definition for lines of code on the SCR form. In addition to SLOC, however, the FDD sometimes reports lines of code without counting any PDL, comments, or blanks. The authors learned that some maintainers had been reporting this number on their SCR forms. Luckily most cases were confined to a single project and a single release. For this release the task leader supplied accurate totals of SLOC added, changed, and deleted.

The tracking of weekly effort is not nearly as thorough and rigorous as the tracking of SCRs. No formal audit process exists to assure that all personnel are submitting WMEFs each week they work on a project. Many managers do try to assure that their personnel submit the forms, but the process is not guaranteed.

Still, we feel confident that the effort data is reasonably complete and accurate. When possible, data validation has been done with the WMEF data. For example, in some cases we found that SCRs had been submitted (after the revised WMEF went into effect) but that no maintainer had listed this SCR on the WMEF. The maintainers who worked on these SCRs were then identified and were required to revise their WMEFs.

6. Conclusions and Future Directions

In this paper, we described descriptive models of a software maintenance environment and an incremental approach for the construction of release productivity models for that environment. The former type of models helped us understand better how and why effort is spent across releases while raising new process improvement issues. The latter type of models helped us provide management tools for maintenance task leaders. In order to validate our approach, a case study was conducted at the NASA Software Engineering Laboratory, where we showed the feasibility of building such models. In addition, we derived a set of lessons learned about our maintenance process which allowed us to propose concrete improvement steps. We would like to emphasize that the models produced in this study are specific to a particular environment. Software organizations seeking such models should not directly apply our models, but instead should construct models specific to their organization by using the process we presented in this paper. Based on these results, some of the many issues that should be further investigated are discussed below.

As more releases are completed, predictive models for the other categories of releases can be developed. Having cost models for all three types of releases, along with an understanding of the outlier subset of high productivity releases, would complete the cost modeling area of our study. Good cost models for the other types of releases might not be obtainable, but further understanding of the overhead of a release might give better guidance on release content.

In addition to the current model, there is a need for an effort prediction model at the change level. This would help the maintainers perform cost/benefit analysis of the change requests and thereby better determine the release content within budget constraints.

The suite of predictive models can also be expanded to include reliability. We would like to be able to predict, for example, the number of errors uncovered during each maintenance release. Such information will lead to more guidance on release content, and to a better understanding of the release testing process.

Acknowledgment

We want to thank Roseanne Tesoriero for her valuable suggestions that helped us improve both the content and the form of this paper.

References

- [Abran & Nguyenkim 1991] Abran, A. and Nguyenkim, H. "Analysis of Maintenance Work Categories Through Measurement," *Proc. Conf. on Software Maintenance 1991*, Sorrento, Italy, pp. 104-113.
- [Basili & Rombach 1988] Basili, V. R. and D. Rombach. "The TAME Project: Towards Improvement-Oriented Software Environments," *IEEE Trans. on Software Engineering*, 14 (6), June 1988, pp. 758-773.
- [Briand et al. 1994] Briand, L., V. R. Basili, Y.-M. Kim and D. Squier. "A Change Analysis Process to Characterize Software Maintenance Projects," *Proc. Int'l. Conf. on Software Maintenance*, Victoria, B. C., Canada, 1994, pp. 38-49.
- [Briand et al. 1995] Briand, L., W. Melo, C. Seaman, and V. Basili. "Characterizing and Assessing a Large-Scale Software Maintenance Organization," *Proc. 17th Int'l. Conf. on Software Engineering*, Seattle, WA, 1995, pp. 133-143.
- [Haziza et al. 1992] Haziza, M., J. F. Voidrot, E. Minor, L. Pofelski and S. Blazy. "Software Maintenance: An Analysis of Industrial Needs and Constraints," *Proc. Conf. on Software Maintenance 1992*, Orlando, Florida, pp. 18-26.
- [Harrison & Cook 1990] Harrison, W. and C. Cook. "Insights on Improving the Maintenance Process Through Software Measurement," *Proc. Conf. on Software Maintenance 1990*, San Diego, CA, pp. 37-45.
- [Hinkle et al. 1995] Hinkle, D. E., W. Wiersma and S. G. Jurs. *Applied Statistics for the Behavioral Sciences*, Boston: Houghton Mifflin, 1995.
- [McGarry et al. 1994] McGarry, F., G. Page, V. R. Basili, and M. Zelkowitz. *An Overview of the Software Engineering Laboratory*, SEL-94-005, December 1994.
- [Paulk et al. 1993] Paulk, M., B. Curtis, M-B Chrissis, C. Weber. "Capability Maturity Model, Version 1.1," *IEEE Software*, July 1993, pp. 18-27.
- [Pigoski & Nelson 1994] Pigoski, T. M. and L. E. Nelson. "Software Maintenance Metrics: A Case Study," *Proc. Int'l. Conf. on Software Maintenance*, Victoria, B.C., Canada, 1994, pp. 392-401.
- [Rombach et al. 1992] Rombach, H., B. Ulery and J. Valett. "Toward Full Life Cycle Control: Adding Maintenance Measurement to the SEL," *J. Systems and Software*, Nov. 1992, pp. 125-138.
- [Schneidewind 1994] Schneidewind, N. "A Methodology for Software Quality: Metrics for Maintenance." Tutorial presented at the Int'l. Conf. on Software Maintenance, Victoria, B. C., Canada, 1994.
- [Valett et al. 1994] Valett, J., S. Condon, L. Briand, Y.-M. Kim and V. Basili. "Building an Experience Factory for Maintenance," *Proc. 19th Annual Software Eng. Workshop*, NASA Goddard Space Flight Center, November 1994.
- [Waligora et al. 1995] Waligora, S., J. Bailey and M. Stark. *Impact of ADA and Object-Oriented Design in the Flight Dynamics Division at Goddard Space Flight Center*, SEL-95-001, 1995.

The Role of Experimentation in Software Engineering: Past, Current, and Future

Victor R. Basili

Institute for Advanced Computer Studies
and
Department of Computer Science
University of Maryland

Abstract

Software engineering needs to follow the model of other physical sciences and develop an experimental paradigm for the field. This paper proposes the approach towards developing an experimental component of such a paradigm. The approach is based upon a quality improvement paradigm that addresses the role of experimentation and process improvement in the context of industrial development. The paper outlines a classification scheme for characterizing such experiments.

1. Introduction

Progress in any discipline depends on our ability to understand the basic units necessary to solve a problem. It involves the building of models¹ of the application domain, e.g., domain specific primitives in the form of specifications and application domain algorithms, and models of the problem solving processes, e.g., what techniques are available for using the models to help address the problems. In order to understand the effects of problem solving on the environment, we need to be able to model various product characteristics, such as reliability, portability, efficiency, as well as model various project characteristics such as cost and schedule. However, the most important thing to understand is the relationship between various process characteristics and product characteristics, e.g., what algorithms produce efficient solutions relevant to certain variables, what development processes produce what product characteristics and under what conditions.

Our problem solving ability evolves over time. The evolution is based upon the encapsulation of experience into models and the validation and verification of those models based upon experimentation, empirical evidence, and reflection. This encapsulation of knowledge allows us to deal with higher levels of abstraction that characterize the

problem and the solution space. What works and doesn't work will evolve over time based upon feedback and learning from applying the ideas and analyzing the results.

This is the approach that has been used in many fields, e.g., physics, medicine, manufacturing. Physics aims at understanding the behavior of the physical universe and divides its researchers into theorists and experimentalists. Physics has progressed because of the interplay between these two groups. Theorists build models to explain the universe - models that predict results of events that can be measured. These models may be based upon theory or data from prior experiments. Experimentalists observe and measure. Some experiments are carried out to test or disprove a theory, some are designed to explore a new domain. But at whatever point the cycle is entered, there is a modeling, experimenting, learning and remodeling pattern.

Science to the early Greeks was observation followed by logical thought. It took Galileo, and his dropping of balls off the tower at Pisa, to demonstrate the value of experimentation. Modern physicists have learned to manipulate the physical universe, e.g. particle physicists. However, physicists cannot change the nature of the universe [8].

Another example is medicine. Here we distinguish between the researcher and the practitioner. Human intelligence was long thought to be centered in the heart. The circulation of the blood throughout the body was a relatively recent discovery. The medical researcher aims at understanding the workings of the human body in order to predict the effects of various procedures and drugs and provide knowledge about human health and well-being. The medical practitioner aims at applying that knowledge by manipulating the body for the purpose of curing it. There is a clear relationship between the two and knowledge is often built by feedback from the practitioner to the researcher.

Medicine began as an art form. Practitioners applied various herbs and curing processes based upon knowledge handed down, often in secret, from generation to generation. Medicine as a field did not really progress, until various forms of learning, based

¹ We use the term model in a general sense to mean a simplified representation of a system or phenomenon; it may or may not be mathematical or even formal.

upon experimentation and model building, took place. Learning from the application of medications and procedures formed a base for evolving our knowledge of the relationship between these solutions and their effects. Experimentation takes on many forms, from controlled experiments to case studies. Depending on the area of interest, data may be hard to acquire. However, our knowledge of the human body has evolved over time. But both grew based upon our understanding of the relationship between the procedures (processes) and its effects on the body (product). The medical practitioner can and does manipulate the body, but the essence of the body, which is physical, does not change. Again, the understanding was based upon model building, experimentation, and learning.

A third and newer example is manufacturing. The goal of manufacturing is to produce a product that meets a set of specifications. The same product is generated, over and over, based upon a set of processes. These processes are based upon models of the problem domain and solution space and the relationship between the two. Here the relationship between process and product characteristics is generally well understood. But since the product is often a man-made artifact, we can improve on the artifact itself, change its essence. Process improvement is performed by experimenting with variations in the process, building models of what occurs, and measuring its effect on the revised product. Models are built with good predictive capabilities based upon a deep understanding of the relationship between process and product.

2. The nature of the software engineering discipline

Like physics, medicine, manufacturing, and many other disciplines, software engineering requires the same high level approach for evolving the knowledge of the discipline; the cycle of model building, experimentation and learning. We cannot rely solely on observation followed by logical thought. *Software engineering is a laboratory science*. It involves an experimental component to test or disprove theories, to explore new domains. We must experiment with techniques to see how and when they really work, to understand their limits, and to understand how to improve them. We must learn from application and improve our understanding.

The researcher's role is to understand the nature of processes and products, and the relationship between them. The practitioner's role is to build "improved" systems, using the knowledge available. Even more than in the other disciplines, these roles are symbiotic. The researcher needs 'laboratories'; they only exist where practitioners build software systems. The practitioner needs to understand how to build

better systems; the researcher can provide the models to make this happen.

Unlike physics and medicine, but like manufacturing, we can change the essence of the product. Our goal is to build improved products. However, unlike manufacturing, software is development not production. We do not re-produce the same object, each product is different from the last. Thus, the mechanisms for model building are different; we do not have lots of data points to provide us with reasonably accurate models for statistical quality control.

Most of the technologies of the discipline are human based. It does not matter how high we raise the level of discourse or the virtual machine, the development of solutions is still based upon individual creativity, and so differences in human ability will always create variations in the studies. This complicates the experimental aspect of the discipline. Unlike physics, the same experiment can provide different results depending on the people involved. This is a problem found in the behavioral sciences.

Besides the human factor, there are a large number of variables that affect the outcome of an experiment. All software is not the same; process is a variable, goals are variable, context is variable. That is, one set of processes might be more effective for achieving certain goals in a particular context, than another set of processes. We have often made the simplifying assumption that all software is the same, i.e., the same models will work independent of the goals, context size, application, etc. But this is no more true than it is for hardware. Building a satellite and a toaster are not the same thing, anymore than developing the micro code for a toaster and the flight dynamic software for the satellite are the same thing.

A result of several of the above observations is that there is a lack of useful models that allow us to reason about the software process, the software product and the relationship between them. Possibly because we have been unable to build reliable, mathematically tractable models, like in physics and manufacturing, we have tended not to build any. And those that we have, are not always sensitive to context. Like medicine, there are times when we need to use heuristics and models based upon simple relationships among variables, even if the relationships cannot be mathematically defined.

3. The available research paradigms

There are various experimental and analytic paradigms used in other disciplines. The analytic paradigms involve proposing a set of axioms, developing a theory, deriving results and, if possible, verifying the results with empirical observations. This is a deductive model which does not require an

experimental design in the statistical sense, but provides an analytic framework for developing models and understanding their boundaries based upon manipulation of the model itself. For example the treatment of programs as mathematical objects and the analysis of the mathematical object or its relationship to the program satisfies the paradigm. Another way of verifying the results is by an existence proof, i.e., the building of a software solution to demonstrate that the theory holds. A software development to demonstrate a theory is different from building a system ad hoc. The latter might be an excellent art form but does not follow a research paradigm.

The experimental paradigms involve an experimental design, observation, data collection and validation on the process or product being studied. We will discuss three experimental models; although they are similar, they tend to emphasize different things.

First we define some terms for discussing experimentation. A *hypothesis* is a tentative assumption made in order to draw out and test its logical or empirical consequence. We define *study* broadly, as an act or operation for the purpose of discovering something unknown or of testing a hypothesis. We will include various forms of experimental, empirical and qualitative studies under this heading. We will use the term *experiment* to mean a study undertaken in which the researcher has control over some of the conditions in which the study takes place and control over (some aspects of) the independent variables being studied. We will use the term *controlled experiment* to mean an experiment in which the subjects are randomly assigned to experimental conditions, the researcher manipulates an independent variable, and the subjects in different experimental conditions are treated similarly with regard to all variables except the independent variable.

The experimental paradigm of physics is epitomized by the scientific method: observe the world, propose a model or a theory of behavior, measure and analyze, validate hypotheses of the model or theory (or invalidate them), and repeat the procedure evolving our knowledge base.

In the area of software engineering this inductive paradigm might best be used when trying to understand the software process, product, people, or environment. It attempts to extract from the world some form of model which tries to explain the underlying phenomena, and evaluate whether the model is truly representative of the phenomenon being observed. It is an approach to model building. An example might be an attempt to understand the way software is being developed by an organization to see if their process model can be abstracted or a tool can be built to automate the process. The model or tool is then applied in an experiment to verify the

hypotheses. Two variations of this inductive approach can be used to emphasize the evolutionary and revolutionary modes of discovery.

The experimental paradigm in manufacturing is exemplified by an evolutionary approach: observe existing solutions, propose better solutions, build/develop, measure and analyze, and repeat the process until no more improvements appear possible.

This evolutionary improvement oriented view assumes one already has models of the software process, product, people and environment and modifies the model or aspects of the model in order to improve the thing being studied. An example might be the study of improvements to methods being used in the development of software or the demonstration that some tool performs better than its predecessor relative to certain characteristics. Note that a crucial part of this method is the need for careful analysis and measurement.

It is also possible for experimentation to be revolutionary, rather than evolutionary, in which case we would begin by proposing a new model, developing statistical/qualitative methods, applying the model to case studies, measuring and analyzing, validating the model and repeating the procedure.

This revolutionary improvement oriented view begins by proposing a new model, not necessarily based upon an existing model, and attempts to study the effects of the process or product suggested by the new model. The idea for the new model is often based upon problems observed in the old model or approach. An example might be the proposal of a new method or tool used to perform software development in a new way. Again, measurement and analysis are crucial to the success of this method.

These approaches serve as a basis for distinguishing research activities from development activities. If one of these paradigms is not being used in some form, the study is most likely not a research project. For example, building a system or tool alone is development and not research. Research involves gaining understanding about how and why a certain type of tool might be useful and by validating that a tool has certain properties or certain effects by carefully designing an experiment to measure the properties or to compare it with alternatives. An experimental method can be used to understand the effects of a particular tool usage in some environment and to validate hypotheses about how software development can best be accomplished.

4. Software engineering model building

A fair amount of research has been conducted in software engineering model building, i.e., people are building technologies, methods, tools, life cycle models, specification languages, etc. Some of the

earliest modeling research centered on the software product, specifically mathematical models of the program function. There has also been some model building of product characteristics, such as reliability models. There has been modeling in the process domain; a variety of notations exist for expressing the process at different levels for different purposes. However, there has not been much experimenting on the part of the model builders: implementation yes, experimentation no. This may in part be because they are the theorists of the discipline and leave it to the experimenters to test their theories. It may in part be because they view their "models" as not needing to be tested - they see them as self-evident.

For example, in defining a notation for abstracting a program, the theorist may find it sufficient to capture the abstraction perfectly, and not wonder whether it can be applied by a practitioner, under what conditions its application is cost effective, what kind of training is needed for its successful use, etc. Similar things might be said about the process modeler.

It may also be that the theorists view their research domain as the whole unit, rather than one component of the discipline. What is sometimes missing is the big picture, i.e., what is the collection of components and how do they fit together? What are the various program abstraction methods and when is each appropriate? For what applications are they not effective? Under what conditions are they most effective? What is the relationship between processes and product? What is the effect of a particular technique on product reliability, given an environment of expert programmers in a new domain, with tight schedule constraints, etc.

One definition of science is the classification of components. We have not sufficiently enumerated or emphasized the roles of different component models, e.g., processes, products, resources, defects, etc., the logical and physical integration of these models, the evaluation and analysis of the models via experimentation, the refinement and tailoring of the models to an application environment, and the access and use of these models in an appropriate fashion, on various types of software projects from an engineering point of view. The majority of software engineering research has been bottom-up, done in isolation. It is the packaging of technology rather than the solving of a problem or the understanding of a primitive of the discipline.

5. What will our future look like?

We need research that helps establish a scientific and engineering basis for the software engineering field. To this end, researchers need to build, analyze and evaluate models of the software processes and products as well as various aspects of the

environment in which the software is being built, e.g. the people, the organization, etc. It is especially important to study the interactions of these models. The goal is to develop the conceptual scientific foundations of software engineering upon which future researchers can build. This is often a process of discovering and validating small but important concepts that can be applied in many different ways and that can be used to build more complex and advanced ideas rather than merely providing a tool or methodology without experimental validation of its underlying assumptions or careful analysis and verification of its properties.

This research should provide the software engineering practitioner with the ability to control and manipulate project solutions based upon the environment and goals set for the project, as well as knowledge based upon empirical and experimental evidence of what works and does not work and when. The practitioner can then rely on a mix of scientific and engineering knowledge and human ingenuity.

But where are the laboratories for software engineering? They can and should be anywhere software is being developed. Software engineering researchers need industry-based laboratories that allow them to observe, build and analyze models. On the other hand, practitioners need to build quality systems productively and profitably, e.g., estimate cost track progress, evaluate quality. The models of process and product generated by researchers should be tailored based upon the data collected within the organization and should be able to continually evolve based upon the organization's evolving experiences. Thus the research and business perspectives of software engineering have a symbiotic relationship. From both perspectives we need a top down experimental, evolutionary framework in which research and development can be logically and physically integrated to produce and take advantage of models of the discipline, that have been evaluated and tailored to the application environment. However, since each such laboratory will only provide local, rather than global, models, we need many experimental laboratories at multiple levels. These will help us generate the basic models and metrics of the business and the science.

This allows us to view our usable knowledge as growing over time and provides some insight into the relationship between software development as an art and as an engineering discipline. As we progress with our deeper understanding of the models and relationships, we can work on harder and harder problems. At the top is always the need to create new ideas, to go where models do not exist. But we can reach these new heights based upon our ability to build on packages of knowledge, not just packages of technologies.

6. Can this be done?

There have been pockets of experimentation in software engineering but there is certainly not a sufficient amount of it [5, 9, 11]. One explicit example, with which the author is intimately familiar, is the work done in the Software Engineering Laboratory at NASA/GSFC [6]. Here the overriding experimental paradigm has been the Quality Improvement Paradigm [1, 4], which combines the evolutionary and revolutionary experimental aspects of the scientific method, tailored to the study of software. The steps of the QIP are:

Characterize the project and environment, i.e., observe and model the existing environment.

Set goals for successful project performance and improvement and organizational learning

Choose the appropriate processes and supporting methods and tools for this project and for study.

Execute the processes, construct the products, collect and validate the prescribed data based upon the goals, and analyze it to provide real-time feedback for corrective action.

Analyze the data to evaluate the current practices, determine problems, record findings, and make recommendations for future project improvements.

Package the experience in the form of updated and refined models and other forms of structured knowledge gained from this and prior projects and save it in an experience base for future projects.

To help create the laboratory environment to benefit both the research and the development aspects of software engineering, the Experience Factory concept was created. The Experience Factory represents a form of laboratory environment for software development where models can be built and provide direct benefit to the projects under study. It represents an organizational structure that supports the QIP by providing support for learning through the accumulation of experience, the building of experience models in an experience base, and the use of this new knowledge and understanding in the current and future project developments [2].

7. The maturing of the experimental discipline

In order to identify patterns in experimental activities in software engineering from the past to the present, I relied on my experience, discussions with the Experimental Software Engineering Group here at the University of Maryland, and some observations in the literature of experimental papers, i.e., papers that reported on studies that were carried out.

This identified some elements and characteristics of the experimental work in software engineering, specifically (1) identification of the components and purposes of the studies, (2) the types and characteristics of the experiments run, and (3) some ideas on how to judge if the field is maturing. These have been formulated as three questions. First, what are the components and goals of the software engineering studies? Second, what kinds of experiments have been performed? Third, how is software engineering experimentation maturing?

7.1. What are the components and goals of the software engineering studies?

Our model for components method is the Goal/Question/Metric (GQM) Goal Template [4]. The GQM method was defined as a mechanism for defining and interpreting a set of operation goals, using measurement. It represents a systematic approach for tailoring and integrating goals with models of the software processes, products and quality perspectives of interest, based upon the specific needs of a project and organization. However, here, we will only use the parameters of a goal to characterize the types of studies performed. There are four parameters: the object of study, the purpose, the focus, and the point of view. A sample goal might be: analyze perspective based reading (object of interest), in order to evaluate (purpose) it with respect to defect detection (focus) from the point of view of quality assurance (point of view). Studies may have more than one goal but the goals are usually related, i.e. there are several focuses of the same object being analyzed or a related set of objects are being studied. In experimental papers, the point of view is usually the researcher trying to gain some knowledge.

object of study: a process, product, or any form of model

purpose: to characterize (what is it?), evaluate (is it good?), predict (can I estimate something in the future?), control (can I manipulate events?), improve (can I improve event?)

focus: the aspect of the object of study that is of interest, e.g., reliability of the product, defect detection/prevention capability of the process, accuracy of the cost model

point of view: the person who benefits from the information, e.g., the researcher in understanding something better

In going through the literature, there appeared to be two patterns of empirical studies, those I will call *human factor* studies, and those that appear to be more broad-based software engineering. The first class includes studies aimed at understanding the human cognitive process, e.g., how individual programmers

perceive or solve problems. The second set of studies appear to be aimed more at understanding how to aid the practitioner, i.e., building models of the software process, product, and their relationship. We will call these *project-based* studies. The reason for making the distinction is that they appear to have different patterns. Many of the human factor studies were done by or with cognitive psychologists who were comfortable with the experimental paradigm. The object of study tended to be small, the purpose was evaluation with respect to some performance measure. The point of view was mostly the researcher, attempting to understand something about programming.

Although the project-based studies are also often from the point of view of the researcher, it is clear that the perspectives are often practitioner based, i.e. the point of view represented by the researcher is that of the organization, the manager, the developer, etc. The object of study is often the software process or product in some form. If we are looking at breadth, there have been an enormous variety of objects studied. The object set which once included only small, specific items, like particular programming language features, has evolved to include entire development processes, like Cleanroom development.

Although the vast majority of such studies are also aimed at evaluation, and a few at prediction; more recently, as the recognition of the complexity of the software domain has grown, there are more studies that simply try to characterize and understand something, like effort distribution, rather than evaluate whether or not it is good.

7.2. What kinds of experiment have been performed?

There are several attributes of an experiment. Consider the following set:

(1) Does the study present results which are descriptive, correlational, cause-effect?

Descriptive: there may be patterns in the data but the relationship among the variables has not been examined

Correlational: the variation in the dependent variable(s) is related to the variation of the independent variable(s)

Cause-effect: the treatment variable(s) is the only possible cause of variation in the dependent variable(s)

Most of the human factor studies were cause-effect. This appears to be a sign of maturity of the experimentalists in that area as well as the size and nature of the problem they were attacking. The project-based studies were dominated by correlational studies early on but have evolved to more descriptive (and qualitative) style studies over time. I believe this

reflects early beliefs that the problem was simpler than it was and some simple combination of metrics could easily explain cost, quality, etc.

(2) Is the study performed on novices or experts or both?

novice: students or individuals not experienced in the study domain

experts: practitioners of the task or people with experience in the study domain

There seems to be no pattern here, except possibly that there are more studies with experts in the project based study set. This is especially true with the qualitative studies of organizations and projects, but also with some of the controlled experiments.

(3) Is the study performed in vivo or in vitro?

In vivo: in the field under normal conditions

In vitro: in the laboratory under controlled conditions

Again, for project-based studies, there appear to be more studies under normal conditions (in vivo).

(4) Is it an experiment or an observational study?

Although the term experiment is often used to be synonymous with controlled experiment, as defined earlier, I have taken a broader definition here. In this view, we distinguish between *experiments*, where at least one treatment or controlled variable exists, and *observational studies* where there are no treatment or controlled variables.

Experiments can be characterized by the number of teams replicating each project and the number of different projects analyzed. As such, it consists of four different experimental classes, as shown in Table 1: blocked subject-project, replicated project, multi-project variation, and a single project. Blocked subject-project and replicated project experiments represent controlled experiments, as defined earlier. Multi-project variation and single project experiments represent what have been called quasi-experiments or pre-experimental designs [7].

In the literature, typically, controlled experiments are in vitro. There is a mix of both novice and expert treatments, most often the former. Sometimes, the novice subjects are used to "debug" the experimental design, which is then run with professional subjects. Also, controlled experiments can generate stronger statistical confidence in the conclusions. A common approach in the blocked subject-project study is the use of fractional factorial designs. Unfortunately, since controlled experiments are expensive and difficult to control if the project is too large, the projects studied tend to be small.

Quasi-experiments can deal with large projects and be easily done in vivo with experts. These experiments tend to involve a qualitative analysis

		# Projects	
		<i>One</i>	<i>More than one</i>
# of Teams per Project	<i>One</i>	Single Project	Multi-Project Variation
	<i>More than one</i>	Replicated Project	Blocked Subject-Project

Table 1: Experiments

component, including at least some form of interviewing.

Observational studies can be characterized by the number of sites included and whether or not a set of study variables are determined a priori, as shown in Table 2. Whether or not a set of study variables are predetermined by the researcher separates the pure qualitative study, (no a priori variables isolated by the observer), from the mix of qualitative and quantitative analysis, where the observer has identified, a priori, a set of variables for observation.

In purely qualitative analysis, deductions are made using non-mathematical formal logic, e.g., verbal propositions [10]. I was only able to find one study that fit in this category and since it involved multiple sites would be classified as a Field Qualitative Study. On the other hand, there are a large number of case studies in the literature and some field studies. Almost all are in vivo with experts and descriptive.

7.3. How is software engineering experimentation maturing?

One sign of maturity in a field is the level of sophistication of the goals of an experiment and its relevance to understanding interesting (e.g., practical) things about the field. For example, a primitive question might be to determine experimentally if various software processes and products could be measured and their characteristics differentiated on the basis of measurement. This is a primitive question but needed to be answered as a first step in the evolution of experimentation. Over time, the questions have become more sophisticated, e.g., Can a change in an existing process produce a measurable effect on the product or environment? Can the measurable characteristics of a process be used to predict the measurable characteristics of the product or

environment, within a particular context? Can we control for product effects, based upon goals, given a particular set of context variables?

Another sign of maturity is to see a pattern of knowledge building from a series of experiments. This reflects the discipline's ability to build on prior work (knowledge, models, experiments). There are various ways of viewing this. We can ask if the study was an isolated event, if it led to other studies that made use of the information obtained from this particular study. We can ask if studies have been replicated under similar or differing conditions. We can ask if this building of knowledge exists in one research group or environment, or has spread to others, i.e., researchers are building on each other's work.

In both these cases we have begun to see progress. Researchers appear to be asking more sophisticated questions, trying to tackle questions about relationships between processes and product characteristics, using more studies in the field than in the controlled laboratory, and combining various experimental classes to build knowledge.

There are several examples of the evolution of knowledge over time, based upon experimentation and learning, within a particular organization or research group. The SEL at NASA/GSFC offers several examples [6]. One particular example is the evolution of the SEL knowledge of the effectiveness of reading related techniques and methods [3]. In fact, inspections, in general, are well studied experimentally.

There is also growing evidence of the results of one research group being used by others. At least one group of researchers have organized explicitly for the purpose of sharing knowledge and experiments. The

		Variable Scope	
		<i>defined a priori</i>	<i>not defined a priori</i>
# of Sites	<i>One</i>	Case Study	Case Qualitative Study
	<i>More than one</i>	Field Study	Field Qualitative Study

Table 2: Observational Studies

group is called ISERN, the International Software Engineering Research Network. Its goal is to share experiences on software engineering experimentation, by experimenting, learning, remodeling and further experimenting to build a body of knowledge, based upon empirical evidence. They have begun replicating experiments, e.g., various forms of replication of the defect-based reading have been performed, and replications of the perspective-based reading experiment are being performed. Experiments are being run to better understanding the parameters of inspection. ISERN has membership in the U.S., Europe, Asia, and Australia representing both industry and academia.

Another sign of progress for experimental software engineering is the new journal by Kluwer, the International Journal of Empirical Software Engineering, whose aim is to provide a forum for researchers and practitioners involved in the empirical study of software engineering. It aims at publishing artifacts and laboratory manuals that support the replication of experiments. It plans to encourage and publish replicated studies, successful and unsuccessful, highlighting what can be learned from them for improving future studies.

Acknowledgments: I would like to thank the members of the Experimental Software Engineering Group at the University of Maryland for their contributions to the ideas in this paper, especially, Filippo Lanubile, Carolyn Seaman, Jyrki Kontio, Walcelio Melo, Yong-Mi Kim, and Giovanni Cantone.

8. References:

- [1] Victor R. Basili, Quantitative Evaluation of Software Methodology, Keynote Address, First Pan Pacific Computer Conference, Melbourne, Australia, September, 1985.
- [2] Victor R. Basili, Software Development: A Paradigm for the Future, COMPSAC '89, Orlando, Florida, pp.471-485, September 1989.
- [3] Victor R. Basili and Scott Green, Software Process Evolution at the SEL, IEEE Software, pp. 58-66, July 1994.
- [4] Victor R. Basili and H. Dieter Rombach, The TAME Project: Towards Improvement-Oriented Software Environments, IEEE Transactions on Software Engineering, vol. 14, no. 6, June 1988.
- [5] V. R. Basili, R. W. Selby, D. H. Hutchens, "Experimentation in Software Engineering," IEEE Transactions on Software Engineering, vol. SE-12, no. 7, pp. 733-743, July 1986.
- [6] Victor Basili, Marvin Zelkowitz, Frank McGarry, Jerry Page, Sharon Waligora, Rose Pajerski, SEL's Software Development Process Improvement Program, IEEE Software Magazine, pp. 83-87, November 1995.
- [7] Campbell, Donald T. and Julian C. Stanley, Experimental and Quasi-experimental Designs for Research, Houghton Mifflin, Boston, MA.
- [8] Lederman, Leon, "The God Particle", Houghton Mifflin, Boston, MA, 1993
- [9] Norman Fenton, Shari Lawrence Pfleeger, and Robert L. Glass, Science and Substance: A Challenge to Software Engineers, IEEE Software, pp. 86 - 94, July 1994.
- [10] A. S. Lee, "A scientific methodology for MIS Case Studies", MIS Quarterly, pp.33-50 March 1989.
- [11] W. L. Tichy, P. Lukowicz, L. Prechelt, and E. A. Heinz, Experimental Evaluation in Computer Science: A Quantitative Study, Journal of Systems and Software, vol. 28, pp. 1.

SIMULATION MODELING OF SOFTWARE DEVELOPMENT PROCESSES

G. F. Calavaro^{1,2}, V. R. Basili², G. Iazeolla¹

¹University of Rome at Tor Vergata
and

²University of Maryland at College Park

iazeolla@info.utovrm.it

ABSTRACT

A simulation modeling approach is proposed for the prediction of software process productivity indices, such as cost and time-to-market, and the sensitivity analysis of such indices to changes in the organization parameters and user requirements.

The approach uses a timed Petri Net and Object Oriented top-down model specification.

Results demonstrate the model representativeness, and its usefulness in verifying process conformance to expectations, and in performing continuous process improvement and optimization.

INTRODUCTION

Reducing the cost of large scale software projects and shortening cycle time, or time to market, is a major goal of most software development organizations.

To pursue such a goal, organizations can set productivity goals for each project, and put in place statistical productivity controls to enable developers and management to take corrective actions when there are deviations from the goal, and to distinguish a random deviation from meaningful deviations.

Simulation is one of the methods for performing such control. It can be used at various points in the software life cycle to perform risk analysis, in terms of time to product, and cost, to verify conformance to expectations, and to perform continuous process improvement and optimization.

This requires that organizations use metrics and models to evaluate and predict effort and

time (Basili 1979), (Fenton 1991).

The intrinsic complexity of the software production process makes it difficult to conduct predictions using strict analytical models. It is often necessary to turn to simulation models to obtain adequate information on the dynamic behavior, the functionality and the performance of the process.

Software development organizations use several models that address the issue of estimating the effort and the time to product delivery.

Existing analytical models, such as the COCOMO model (Bohem 1981), the Mark II Function Point model (C.R. Symons, 1988) etc., generally only provide predictions of total development effort, without any information about how these are distributed throughout the process. Existing simulation models also share this deficiency. In both cases, no information is given on the instantaneous dynamic behavior of effort versus time, and on the effect of changing user requirements during development time.

Field experiences show that requirements and available resources change during development, so cost and delivery time change as well. Therefore, it is very important to have models which predict the dynamic behavior of cost and time while requirements change.

To reach this goal, this paper proposes a simulation approach to evaluate the effort spent over time by each activity of the process, the estimated delivery time, and the size of the final product.

As illustrated in Figure 1, the approach consists of a process simulation model that provides the behavior over time of quantities such as:

- *prod_size* : the measure of the delivered code size (in lines of code);
- *work_e* : the development work effort (in

Work partially supported by the National MURST project "Performability V&V of Products and Processes in Software Engineering", the CNR project "CE and EF in the Software process", the CERTIA Project "Software Performance V&V", and by NASA grant NSG-5123.

person-weeks);
- *delivery_t*: the time to product (in weeks);
as a function of the user requirement size (in function-points), and the organization parameters.

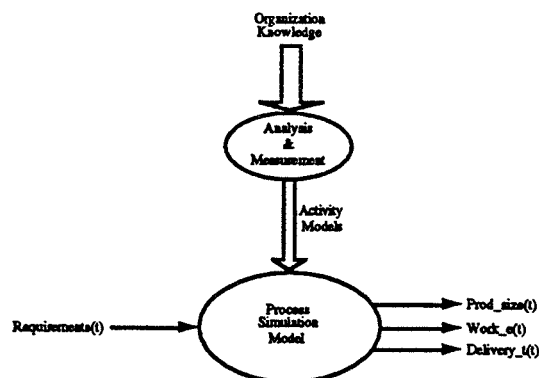


Figure 1: The simulation modeling approach

The model is parametrized on the basis of measurements and analyses of data coming from knowledge of the simulated organization.

The second Section of the paper describes the basic assumptions of this work. The third presents the considered model. The fourth presents the models of the process activities. The fifth Section synthetically illustrates the simulation model, and the model for requirements generation. The sixth presents the results of the simulation experiments.

MAIN ASSUMPTIONS

This work assumes the development process is split into a sequence of *activities* according to the Waterfall model.

An *artifact* is defined as any kind of document, paper, or file produced or used by an activity of the development process.

For each activity it is assumed there is an *input artifact* and an *output artifact*. Input artifact is the artifact the activity uses as an information source to produce the output artifact.

Example artifacts are: the requirement specification document, the requirement analysis document, the architectural design document, the detailed design document, the code after implementation, the code after system test, the code after acceptance test.

It is also assumed that:

- There exist metrics to express the size of any artifact.
- There exist models for the estimation of the output artifact size as a function of the input

artifact size. For example, the detailed design document size can be estimated on the basis of the architectural design document size.

- There exist models to express the amount of resources each activity requires to produce the output artifact on the basis of the input artifact size.

THE CONSIDERED MODEL

The considered process model is illustrated in Figure 2. It includes a model of standard software development process activities such as: Requirement Analysis (ReqAna), Preliminary or architectural Design (PreDes), Detailed Design (DetDes), Implementation (Impl), System Test (SysTest), and Acceptance Test (AccTest).

The input variable of each activity is the estimated size of the output artifact produced by the previous activity. The output variable of each activity is the estimated size of its output artifact and an estimated measure of the activity effort. The latter estimation is send to a *data collector* for recording and evaluating purpose (see later).

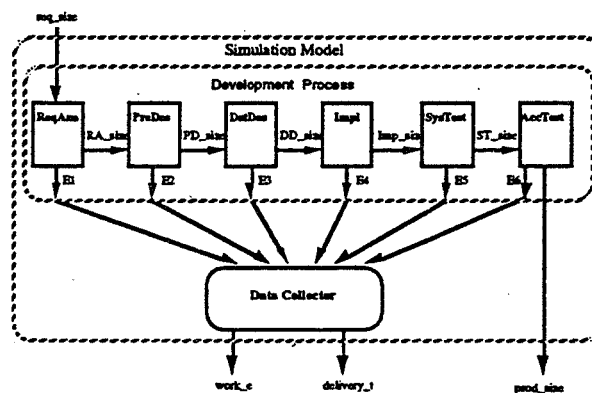


Figure 2. The considered software process model

Input and output variables dynamically change over time. In other words, each activity takes new input values at each time instant and yields the corresponding output values. There obviously exists a time delay in producing outputs, which is dealt with by the activity model.

As shown in Figure 2, the *ReqAna* block receives as input the size of the Requirement Document, *req_size*, and produces as output *RA_size* (the estimated size of the Specification Document produced by the Requirement Analysis activity), besides the measure of the

ReqAna effort ($E1$).

In a similar way, each of the following blocks receive measures of input artifact size, and yield measures of estimated output artifact size, for the following block, and of required effort for the *Data Collector* block.

The final block, *AccTest* block, receives as input the size of the code after System Test, ST_size , and produces as output the size of the final product, $prod_size$, with the measure of the *AccTest* effort ($E6$).

The *Data Collector* block obtains the $E1$ through $E6$ effort values and yields the total time integral of such values, $work_e$, in addition to the delivery time of the final product, $delivery_t$.

THE ACTIVITY BLOCK MODELS

This Section illustrates the details of the internal behavior of each standard activity block. Such behavior is expressed in terms of an input/output function that transforms the input artifact size into the output artifact size and into a measure of the required effort.

For the sake of conciseness the Rayleigh function is assumed as the basic activity model. Such a function can obviously be replaced by any empirically derived function, in case this is believed to better represent the organization's behavior.

The use of the Rayleigh function is supported by the large amount of literature assessing its usefulness as a good model of the software development process, (Putnam 1978), (Fenton 1991), by marketed prediction tool products as SLIM, and by the fact that empirical functions derived by large organizations, such as NASA, are very similar to the Rayleigh function, as shown in Figure 3, derived from (SEL-81-305).

As seen from the Figure, the behavior of effort versus time for each individual activity follows very closely the Rayleigh function, mathematically expressed by:

$$E(t) = \frac{W(t)}{T(t)^2} t e^{-t^2/2T(t)^2} \quad (1)$$

where $E(t)$ is the instantaneous effort required at time t (the equivalent of full time staffing level), $W(t)$ is the estimated total effort of the activity (the integral of $E(t)$ over time) expressed in person-week at time t , and $T(t)$ is the estimated delivery time (weeks) of the activity for the

artifact at time t .

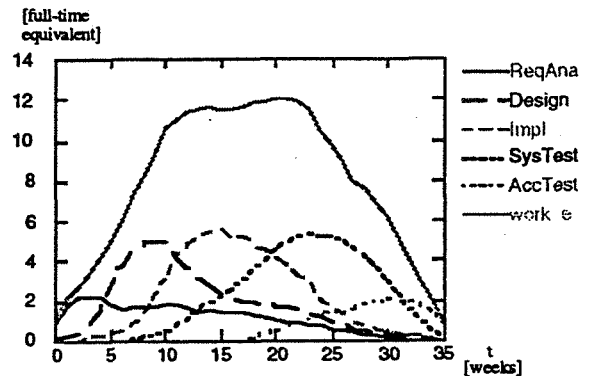


Figure 3. Example of individual activity effort and total life-cycle effort for the NASA software development process

Quantities $W(t)$ and $T(t)$ vary with time t , since it is assumed that the original input, req_size , changes over time. Their values are assumed to be given by conventional models, inspired by the COCOMO equations (SEL-81-305), as follows:

$$out_size(t) = a_1 in_size(t)^{b_1} + c_1$$

$$W(t) = a_2 out_size(t)^{b_2} + c_2 \quad (2)$$

$$T(t) = a_3 W(t)^{b_3} + c_3$$

where $in_size(t)$ is the input artifact size for the generic activity, and $out_size(t)$ is the output artifact size, at time t .

According to the Putnam assumption (Putnam 78), this work also assumes that, for each activity, the outcome $out_size(t)$ takes place only when the instantaneous value of $E(t)$ reaches its peak.

The effort spent after the peak (the down sloping part of various curves in Figure 3) is for rework and updates due to requirements changes. The overlaps between down and upward sloping of curves relating to contiguous activities, implicitly and synthetically represent iterations and interactions among teams. The corresponding effort values are thus implicitly considered in the model. This concept is further refined in next Section.

Parameters a_i , b_i , and c_i , for each activity, are assumed to be derived from empirical data

from the modeled organization.

THE SIMULATION MODEL

In literature there are few examples of simulation models of software development processes. Examples of such models are the *Articulator* by Scacchi and Mi (Scacchi and Mi 1993) and *System Dynamics* by Abdel-Hamid and Madnik (Abdel-Hamid and Madnik 1991).

The Scacchi work deals with a knowledge-based computing environment for modeling, analyzing and simulating complex organizational processes. Its purpose is to simulate the organizational behavior in terms of agents, tasks, and resource allocation.

The Abdel-Hamid work deals with the simulation of software development organizations based on system dynamics techniques.

Neither the former, nor the latter deal with the simulation of the effects of the requirements changes on product costs and time to delivery, which is the subject of this paper.

For the sake of conciseness, the details of the simulation model used in the illustrated approach are given elsewhere (Calavaro et al. 1995).

The model replicates the process scheme illustrated in Figure 2.

An object oriented approach is used to specify the main objects of the simulator, and their connections, and a timed Petri Net, top-down hierarchical approach is used to specify the dynamics of the simulation model and the data flows among objects (Calavaro 1995).

The combination of such approaches in the model specification, makes the simulator easily implementable and adaptable to various process organizations.

Any implementation language can be used. However, languages which are specifically meant for dynamic system simulation, such as DYNAMO and SIMNON, are preferred. For this paper, the latter has been used, since it is particularly oriented to non-linear system analysis.

Model Input Generation

In most real systems, generation of user requirements is usually performed by the development organization in conjunction with the customers, and yields the requirements of the system to be developed.

In the simulation model the requirements size (*req_size*) is the model input. This dynamically

changes over time. The model assumes that the generation activity is external to the process activities, and so its effort is not part of the effort calculations.

The *req_size* value is assumed to be expressed in number of Function Points (FP). The input is assumed to start at time 0 of the process dynamics.

For the experiment in this paper *req_size* is expressed by the following equation:

$$\text{req_size}(t) = a_4 (1 - e^{-t/b_4}) + c_4 \quad (3)$$

where c_4 is the initial requirement size, a_4 is the changed requirement size, and b_4 is a time constant. According to this expression, the requirements increase along time by a negative exponential rate, and reach their stable value $a_4 + c_4$ asymptotically in time. Other equations could be used as well.

The model assumes $a_4 = 50$ [FP], $c_4 = 100$ [FP], and $b_4 = 15$ [weeks].

In other words, it assumes a 50% increase in the requirements size during the life cycle.

SIMULATION RESULTS

Obtained results are expressed by curves that give the effort values over time for each process activity, the global process effort, and the time to product.

Figure 4 shows the simulated effort density for various process activities (ReqAna, PreDes, DetDes, Impl, SysTest, and AccTest) and for the total modeled process (*work_e*).

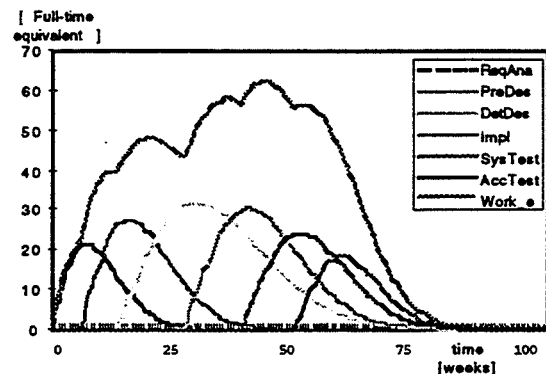


Figure 4. Simulated effort density for various process activities and for the total modeled process

The simulated total effort density for the

process, *work_e*, is the sum of the individual activities' efforts. Its integral gives *work_e* = 4584 person_weeks for a *prod_size* = 103 KLOCs, in a *delivery_t* = 70 weeks.

The curves in Figure 4 are the results of our simulation. They are qualitatively similar to those in Figure 3, which are empirically derived.

This supports the validity of the proposed simulation model. The introduction of realistic values for the *a_i*, *b_i*, and *c_i* parameters is the only requirement to obtain model validation on a quantitative basis.

The valleys in the top of Figure 4 *work_e* curve are a consequence of the assumption that each activity starts when the previous activity effort reaches its peak. Such valleys are not evident in the empirical NASA curve, seen in Figure 3. We believe that this difference is at least partly due to the fact that in any real environment the activity starts a little bit before the previous activity peak, since unofficial artifacts are delivered to the following activity before the official ones are ready.

This shows the representativeness of the proposed approach, for use in Waterfall-like organizations, for the prediction of the software production costs and delivery times, as well as for the analysis of the sensitivity of costs and times to changes in organization parameters, and to the variation in user requirements.

The top-down hierarchical model specification permits adaptation of the proposed simulator to non-waterfall various process models.

CONCLUSIONS and FUTURE RESEARCH

Simulation is one of the productivity control methods that enables software developers and managers to take corrective actions and perform risk analysis, in terms of time to product and cost, to verify conformance to expectations, and to perform continuous process improvement and optimization.

This paper has introduced a software process simulation modeling approach for the prediction of the software production costs and delivery times, and analysis of sensitivity to the changes in organization parameters, and user requirements.

The model is based on a top-down hierarchical model specification that can be used to adapt the proposed simulator to various

process models.

This is part of future research, with the explicit modeling of the interactions between process activities and of the modeling of product iterations.

Future research also includes explicitly representing physical factors and agents that characterize various process activities, by the use of specialized software process languages.

REFERENCES

- Abdel-Hamid T.K. Madnick S.E., 1991. *Software Project Dynamics - An Integrated Approach*. Prentice Hall, Englewood Cliffs, NJ.
- Basili V.R., 1989. "Software Development: A Paradigm for the Future" In *Proc. 13th Int'l Computer Software and Applications Conf.*, (Orlando, FL, Sept.), CS Press, 471-485.
- Bohem B. W., 1981. *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ.
- Calavaro G.F., 1995. *Experience Factory and Concurrent Engineering for Software Process Optimization*, Ph.D. Dissertation, University of Rome "Tor Vergata", Rome, Italy.
- Calavaro G.F., Basili V.R., Iazeolla G., 1995. "Simulation Modeling of Software Development Processes", *Technical Report University of Rome "Tor Vergata"*, RI.95.06, Rome, Italy.
- Fenton N.E., 1991. *Software Metrics, A rigorous approach* Chapman & Hall, London, UK.
- Putnam L.H., 1978. "A General Empirical Solution to the Macro Software Sizing and Estimating Problem", *IEEE Transaction on Software Engineering*, (July) 345-361.
- Scacchi W., Mi P., 1993. "Modeling, Integrating, and Enacting Software Production Processes" In *Proc. 3rd Irvine Software Symposium* (Costa Mesa, CA, April)
- Symons C.R., 1988. "Function Point Analysis: Difficulties and Improvements", *IEEE Trans. on Software Engineering*, (14):1, (January) 2-11
- SEL-81-305, 1992. "Recommended Approach to Software Development", Revision 3, *Software Engineering Laboratory series*, SEL-81-305, NASA-GSFC, Greenbelt, MD.

omit
THIS
PAGE

SECTION 3—TECHNOLOGY EVALUATION

The technical papers included in this section were originally prepared as indicated below.

- *Qualitative Analysis for Maintenance Process Assessment*, L. Briand, Y. Kim, W. Melo, C. B. Seaman, and V. R. Basili., University of Maryland, Computer Science Technical Report, CS-TR-3592, UMIACS-TR-96-7, January 1996
- *Evolving and Packaging Reading Technologies*, V. R. Basili, *Proceedings of the Third International Conference on Achieving Quality in Software*, Florence, Italy, January 1996
- *The Empirical Investigation of Perspective-Based Reading*, V. R. Basili, S. Green, O. Laitenberger, F. Shull, S. Sorumgaard, M. Zelkowitz, University of Maryland, Computer Science Technical Report, CS-TR-3585, UMIACS-TR-95-127, December 1995
- "The Impact of Ada and Object-Oriented Design in NASA Goddard's Flight Dynamics Division," S. Waligora , J. Bailey, and M. Stark, *Proceedings of the 13th Annual Washington Ada Symposium (WAdaS96)*, July 1996

Qualitative Analysis for Maintenance Process Assessment

Lionel Briand
CRIM
Montréal, Québec, Canada

Yong-Mi Kim
Walcélio Melo
Carolyn Seaman
Victor Basili
Institute for Advanced Computer Studies
University of Maryland
College Park, MD, USA

56-61

415772

360848

38A.

Abstract

In order to improve software maintenance processes, we first need to be able to characterize and assess them. These tasks must be performed in depth and with objectivity since the problems are complex. One approach is to set up a measurement-based software process improvement program specifically aimed at maintenance. However, establishing a measurement program requires that one understands the problems to be addressed by the measurement program and is able to characterize the maintenance environment and processes in order to collect suitable and cost-effective data. Also, enacting such a program and getting usable data sets takes time. A short term substitute is therefore needed.

We propose in this paper a characterization process aimed specifically at maintenance and based on a general qualitative analysis methodology. This process is rigorously defined in order to be repeatable and usable by people who are not acquainted with such analysis procedures. A basic feature of our approach is that actual implemented software changes are analyzed in order to understand the flaws in the maintenance process. Guidelines are provided and a case study is shown that demonstrates the usefulness of the approach.

This work was supported by NASA grant NSG-5123, NSF grant 01-5-24845, and by NSERC, Canada.

E-mails: {basili | kimy | melo | cseaman }@cs.umd.edu and lbriand@crim.ca

1. Introduction

During the past few years the definition and improvement of software processes has been playing an increasingly prominent part in software development and maintenance. The improvement of software maintenance processes is of particular interest because of the length of time spent in maintenance during the software life cycle, and the ensuing lifetime costs, as well as the large number of legacy systems still being maintained. Improvement requires building an understanding of what is actually happening in a project (the term "project" here refers to the continuous maintenance of a given system), in conjunction with building a measurement program.

Establishing a measurement program integrated into the maintenance process is likely to help any organization achieve an in-depth understanding of its specific maintenance issues and thereby lay a solid foundation for maintenance process improvement [RUV92]. However, defining and enacting a measurement program takes time. A short term, quickly operational substitute is needed in order to obtain a first quick insight, at low cost, into the issues to be addressed. Furthermore, defining efficient and useful measurement procedures first requires a characterization of the maintenance environment in which measurement takes place, such as organization structures, processes, issues, and risks [BR88].

Part of this characterization is the identification and assessment of issues that must be addressed in order to improve the quality and productivity of maintenance projects. Because of the complexity of the phenomena studied, this is a difficult task for the maintenance organization. Each project may encounter specific difficulties and situations that are not necessarily alike across all the organization's maintenance projects. This may be due in part to variations in application domain, size, change frequency, and/or schedule and budget constraints. As a consequence, each project must first be analyzed as a separate entity even if, later on, commonalities across projects may require similar solutions for improvement. Informally interviewing the people involved in the maintenance process would be unlikely to accurately determine the real issues. Maintainers, users and owners would likely each give very different, and often contradictory, insights on the issues due to their biased or incomplete perspectives.

This paper presents a qualitative and inductive analysis methodology for performing objective characterizations and assessments that addresses both the understanding and measurement aspects of improvement. It encompasses a set of procedures which aids the determination of causal links between maintenance problems and flaws in the maintenance organization and process. Thus, a set of concrete steps for maintenance quality and productivity improvement can

be taken based on a tangible understanding of the relevant maintenance issues. Moreover, this understanding provides a solid basis on which to define relevant software maintenance models and metrics.

Section 2 gives an overview of the basic steps of the proposed assessment methodology, and a discussion of the supporting technologies and capabilities it requires. Section 3 presents the supporting technologies we actually used to execute the assessment method. Section 4 presents each step of the assessment process in detail by going through a case study. The experience we gained conducting this case study is the source for much of the guidance we offer in this paper, as well as the basis for the fine-tuning of the assessment methodology itself. Section 5 describes the next logical step after a qualitative assessment, the design of a measurement program aimed at quantitatively monitoring and improving software maintenance. Section 6 outlines the main conclusions of this experience and the future research directions.

2. Overview of the Maintenance Assessment Method

We present below a general description of the maintenance assessment method and the capabilities it requires. Maintenance is defined here as any kind of enhancement, adaptation or correction performed on an operational software system. At the highest level of abstraction, parts of the assessment process are not specific to maintenance and could be used for development. However, the taxonomies and guidelines developed to support this process and presented in Section 3 are specifically aimed at maintenance.

2.1 The steps of the method

We propose a qualitative and inductive methodology in order to characterize and assess software maintenance processes and organizations and thereby identify their specific problems and needs. This methodology encompasses a set of procedures which attempt to determine causal links between maintenance problems and flaws in the maintenance organization and process. This allows for a set of concrete steps to be taken for maintenance quality and productivity improvement, based on a tangible understanding of the relevant maintenance issues in a particular maintenance environment. The steps of this methodology are summarized as follows:

Step 1: Identify the organizational entities with which the maintenance team interacts and the organizational structure in which maintainers operate. In this step the distinct teams, working groups, and their roles in the change process are identified. Information flows between actors are also determined.

- Step 2:** Identify the phases involved in the creation of a new system release. As opposed to the notion of activity, defined below, phases produce one or several intermediate or final release products which are reviewed according to quality assurance procedures, when they exist, and are officially approved. In addition, the phases of a release are ordered in time, although they may be somewhat overlapping, and are clearly separated by milestones. Software artifacts produced and consumed by each phase must be identified. Actors responsible for producing and validating the output artifacts of each phase have to be identified and located in the organizational structure defined in Step 1.
- Step 3:** Identify the generic activities involved in each phase, i.e., decompose life-cycle phases to a lower level of granularity. Identify, for each low-level activity, its inputs and outputs and the actors responsible for them.
- Step 4:** Select one or several representative past releases for analysis in order to better understand process and organization flaws.
- Step 5:** Based in part on release documents and error report forms, analyze the problems that occurred while performing the software changes in the selected releases in order to produce a causal analysis document. The knowledge and understanding acquired through steps 1-3 are necessary in order to understand, interpret and formalize the information described in the causal analysis document.
- Step 6:** Establish the frequency and consequences of problems due to flaws in the organizational structure and the maintenance process by analyzing the information gathered in Step 5.

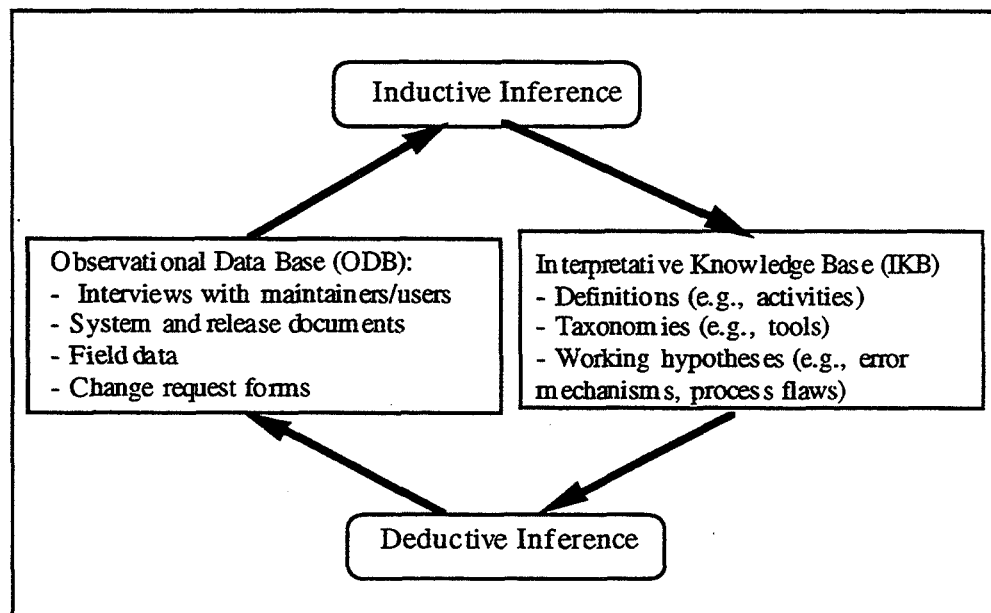


Figure 1: Qualitative Analysis Process for Software Maintenance

This process is essentially an instantiation of the generic qualitative analysis process defined in [SS92]. Figure 1 illustrates at a high level our maintenance-specific qualitative analysis process. It is a combination of both inductive and deductive inferences. The collected information, such as field notes or interviews, comprise the *Observational Database* (ODB), while the models built using the information from the ODB goes into the *Interpretative Knowledge Base* (IKB). Inductive inferences are made from the collected information. Deductive inferences occur when, based on our IKB, we derive expectations about the real world. An example of such an expectation might be that all errors can be exhaustively classified according to defined taxonomies. When comparing these expectations with new field information feeding the ODB, we can experimentally validate and refine our taxonomies, process models, organizational models and working hypotheses (all in the IKB). Then the data collection process is refined in order to solve ambiguities and answer new questions, which leads to refined and revised inductive inferences. The process continues in an iterative fashion. This iterative pattern not only applies to the overall assessment process, but also to several of the individual steps. For example, in our case study, performing Step 1 revealed additional issues to be addressed in building an organizational model, and led to the selection and use of a more sophisticated modeling approach. The iterative nature of these steps is described in more detail in Section 4.

2.2 Capabilities required

During the case study, we were faced with several tasks which required supporting pieces of technology. In this section, we describe the requirements of these technologies, which could be satisfied in a number of different ways. In section 3, we describe in detail the representations and taxonomies we chose to satisfy these requirements during our case study.

2.2.1 Step 1: Organizational Modeling

The first technology we needed was a representation in which to build the organization model (the first step of the assessment method). We identified the following requirements for an optimal organizational modeling approach:

Req1: The modeling methodology had to facilitate the detection of conflicts between organizational structures and goals. For example, inconsistencies between the expectations and intentions of interfacing actors seemed to be a promising area of investigation.

- Req2:** We needed to capture many different types of relationships between actors. These included relationships that contributed to information flow, work flow, and fulfillment of goals. The explicit and comprehensive modeling of all types of relationships was necessary in this context and we believe it is likely to be relevant in other environments as well.
- Req3:** Different types of organizational entities had to be captured: individuals, their official position in the organizational structure, and their roles and activities in the maintenance process. It was important not only to be able to model at different levels of detail, but also to provide different views of the organization, each relaying different information.
- Req4:** Links between the organization and the maintenance process model had to be represented explicitly.
- Req5:** The notation had to aid in communication through intuitive concepts and graphical representation.
- Req6:** We had to be able to flexibly capture information about the maintenance working environment, e.g., available tools and methods.

The process modeling literature provides many examples of techniques for representing various aspects of process. Process modeling is performed for a number of different purposes, including process analysis and process improvement. The "process" under study here includes all the activities involved in the development (and sometimes maintenance) of software. The representations we considered, and the one we finally chose, are described in sections 3.1 and 3.2.

2.2.2 Steps 2 and 3: Process Modeling

Another technology required is a way to model relevant aspects of the maintenance process. However, we found that once the above requirements have been satisfied by the organizational model, the process "model" does not need to be sophisticated. All that is required is a breakdown of the process into its constituent parts, and identification of the inputs and outputs (artifacts) of each part. The classification schemes we used are described in section 3.3.

2.2.3 Step 5: Causal Analysis

The other technologies required to implement the assessment method take the form of descriptions of different entities that are specific to the environment being studied. For example, the causal analysis step requires a list, or taxonomy, of types of maintenance flaws that take place in the environment. Also required by the causal analysis step is a data collection guide which

describes, in terms tailored to the studied environment, the information that needs to be collected in this step. These taxonomies and guides are described in section 3.4.

3. Technologies Used

This section describes the technologies used by the authors to satisfy the requirements listed in the last section, to conduct the case study described in section 4. The pieces of technology described here are not the only possible choices, but they represented the best options for our circumstances and environment.

3.1 An organizational modeling representation

During the case study, it was clear that the organizational model built in step 1 would be central to the assessment. Thus, we were faced with the crucial task of finding an appropriate organizational modeling approach. We first looked in the process literature for such a technology. Representation of organizational structure in most process work is limited to the representation of roles [BK94, LR93], with a few notable exceptions. One is the approach presented in [K91], which uses Statemate. Statemate models include three perspectives, one of which is the organizational perspective, which identifies the people who participate in the process and the information channels between them. Role Interaction Networks (RINs) [R92] (implemented in the modeling tool Deva [MCC92]) also describe process participants and the information that passes between them. The use of CRC cards in the Pasteur tool [CC93] completely describes a process in terms of the process participants and their work dependencies. None of these approaches, however, provides the ability to represent the richness of types of persons, groups, relationships, and structures that we require, e.g., conflicting objectives, synergy between objectives, risk management mechanisms. Furthermore, these approaches do not provide straightforward links between quantitative data and organizational models. This issue is important, as we shall see in Section 5, because the organizational model will be an important tool in the eventual quantitative analysis of the maintenance process.

Yu's Actor-Dependency (A-D) model [YM94] is another modeling notation that shares some of the characteristics of the three described above. In particular, A-D models are based on process participants and the relationships between them, including information flow relationships. These relationships are not limited, however, to information flows. In addition, A-D models provide a variety of ways to represent members of the organization that we believe to be based on a clear and convenient paradigm (see section 3.1.2). Like the above approaches, A-D models were not designed originally to facilitate quantitative data collection and analysis. However, in practice,

we observed that the A-D model was easily modified for this purpose. In Section 5, we describe some of these modifications and in [B+95] we provided a more detailed list of enhancements we proposed to the A-D model. Because of the richness of its underlying paradigm, this modeling approach has been chosen as the representation for our organization model. A-D models are described in more detail in the sections below.

This modeling language provides a basic organizational model with several enhancements, only one of which we will describe here. The basic Actor-Dependency model represents an organizational structure as a network of dependencies among organizational entities, or actors. The enhancement which we have used, called the Agent-Role-Position (ARP) model, provides a useful decomposition of the actors themselves. These two representations are described briefly in the following sections. For a more detailed description, see [YM93].

3.1.1. The basic Actor-Dependency (AD) model

In this model, an organization is described as a network of interdependencies among active organizational entities, i.e., actors. A node in such a network represents an organizational actor, and a link indicates a dependency between two actors. Examples of actors are: someone who inspects units, a project manager, or the person who gives authorization for final shipment. Documents to be produced, goals to be achieved, and tasks to be performed are examples of dependencies between actors. When an actor, A1, depends on A2, through a dependency D1, it means that A1 cannot achieve, or cannot efficiently achieve, its goals if A2 is not able or willing to fulfill its commitment to D1. The AD model provides four types of dependencies between actors:

- In a *goal dependency*, an actor (the depender) depends on another actor (the dependee) to achieve a certain goal or state, or fulfill a certain condition (the dependum). The depender does not specify how the dependee should do this. A fully built configuration, a completed quality assessment, or 90% test coverage of a software component might be examples of goal dependencies if no specific procedures are provided to the dependee(s).
- In a *task dependency*, the depender relies on the dependee to perform some task. This is very similar to a goal dependency, except that the depender specifies how the task is to be performed by the dependee, without making the goal to be achieved by the task explicit. Unit inspections are examples of task dependencies if specific standard procedures are to be followed.

- In a *resource dependency*, the depender relies on the dependee for the availability of an entity (physical or informational). Software artifacts (e.g. designs, source code, binary code), software tools, documents, and any kind of computational resources are examples of resource dependencies.
- A *soft-goal dependency* is similar to a goal dependency, except that the goal to be achieved is not sharply defined, but requires clarification between depender and dependee. The criteria used to judge whether or not the goal has been achieved is uncertain. Soft-goals are used to capture informal concepts which cannot be expressed as precisely defined conditions, as are goal dependencies. High product quality, user-friendliness, and user satisfaction are common examples of soft-goals because in most environments, they are not precisely defined.

Three different categories of dependencies can be established based on degree of criticality:

- *Open dependency*: the depender's goals should not be significantly affected if the dependee does not fulfill his or her commitment.
- *Committed dependency*: some planned course of action, related to some goal(s) of the depender, will fail if the dependee fails to provide what he or she has committed to.
- *Critical dependency*: failure of the dependee to fulfill his or her commitment would result in the failure of all known courses of action towards the achievement of some goal(s) of the depender.

The concepts of open, committed, and critical dependencies can be used to help understand actors' vulnerabilities and associated risks. In addition, we can identify ways in which actors alleviate this risk. A commitment is said to be:

- *Enforceable* if the depender can cause some goal of the dependee to fail.
- *Assured* if there is evidence that the dependee has an interest in delivering the dependum.
- *Insured* if the depender can find alternative ways to have his or her dependum delivered.

In summary, a dependency is characterized by three attributes: type, level of criticality, and its associated risk-management mechanisms. The type (resource, soft-goal, goal, and task) represents the issue captured by the dependency, while the level of criticality indicates how

important the dependency is to the depender. Risk-management mechanisms allow the depender to reduce the vulnerability associated with a dependency.

Figure 2 shows a simple example of an AD model. A Manager oversees a Tester and a Developer. The Manager depends on the Tester to efficiently and effectively test the product. This is a task dependency because there is a defined set of procedures that the Tester must follow. In contrast, the Manager also depends on the Developer to develop, but the Developer has complete freedom to follow whatever process he or she wishes, so this is expressed as a goal dependency. Both the Tester and the Developer depend on the Manager for positive evaluations, where there are specific criteria to define "positive", thus these are goal dependencies. The Tester depends on the Developer to provide the code to be tested (a resource), while the Developer depends on the Tester to test the code well (good coverage). Assuming that there are no defined criteria for "good" coverage, this is a soft-goal dependency.

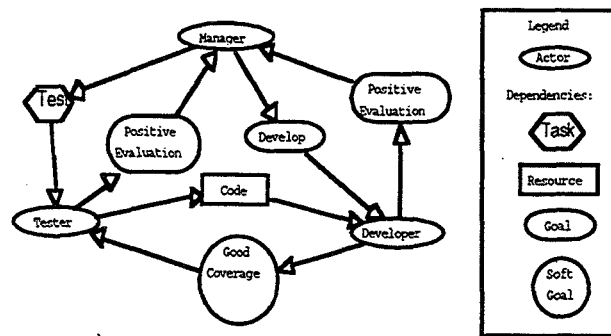


Figure 2: A simple example of an AD model

3.1.2. The Agent-Role-Position (ARP) decomposition

In the previous section, what we referred to as an actor is in fact a composite notion that can be refined in several ways to provide different views of the organization. *Agents*, *roles*, and *positions* are three possible specializations of the notion of actor which are related as follows:

- An agent occupies one or more positions
- An agent plays one or more roles.
- A position can cover different roles in different contexts

Figure 3 shows an example of an actor decomposition. These three types of specialization are useful in several ways. They can be used to represent the organization at different levels of

detail. Positions provide a high-level view of the organization whereas roles provide more details. The use of agents allows the modeler to go even further and specify specific individuals. In addition, the ARP decomposition could be especially useful when extending the use of AD models to quantitative analysis, as explained in Section 5.

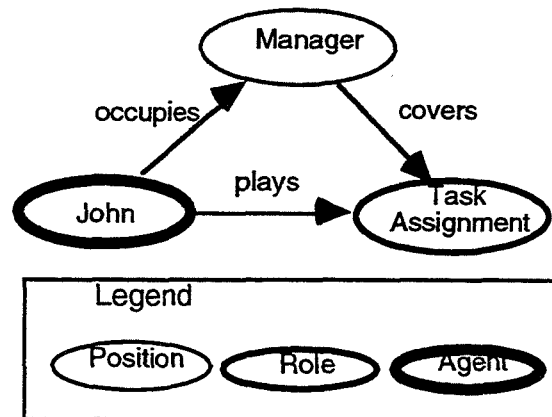


Figure 3. Associated Agent, Position, and Role

3.1.3. Limitations of the organizational model

The AD modeling method satisfied, at least partially, all of the modeling requirements presented in section 2.2, except requirement Req6. However, there were some difficulties. Fulfillment of Req1 and Req2 was impeded by the difficulty of distinguishing between task, goal, resource, and soft-goal dependencies, and between critical, committed, and open dependencies. These categories did not always adequately describe the dependencies that arose in our model. In addition, although the notions of enforcement, assurance and insurance helped to satisfy requirement Req5, they are difficult to represent explicitly in the AD model representation. For more details on that subject, see [B+95].

3.1.4. Value of the organizational model

Modeling the organizational context of the maintenance process was a very important step in the maintenance analysis process. A model of the organization was necessary for communication with maintenance process participants. Gathering organizational information and building the model was critical to our understanding of the work environment and differences across projects. The model was also useful in checking the consistency and completeness of the maintenance process model. For example, the organizational model allowed us to determine whether or not all roles in the process model were assigned to actors in the organization.

In addition, we found that several actor decomposition patterns that were of particular interest in identifying potential organizational problems:

- unassigned roles: nobody has official responsibility for a given role and its associated activities.
- numerous roles associated with a position: the position may be overloaded and/or incompatible roles may be played by one position.
- shared roles across positions: can the responsibility be shared or is this an opportunity for confusion?
- variations of role-position associations across maintenance projects: is this variation due to a lack of definition or to necessary adjustments from project to project?

3.2 Taxonomy of maintenance methods and tools

It was not possible to capture attributes of the maintenance working environment (i.e. tools and methods) within an AD model (requirement Req6). For our case study, we collected this information during step 1 but kept it separate from the organizational model. To organize this information, we found it useful to develop a taxonomy of tools and methods which are relevant in the modeled environment. Figure 4 shows a taxonomy that we think is a good characterization of the information to be gathered about the maintenance environment under study. The taxonomy shows only the first level of abstraction, so that it can be specialized for a particular maintenance environment.

Maintenance tools:
<ul style="list-style-type: none"> • impact analysis and planning tool • tools for automated extraction and representation of control and data flows • debugger • generator of cross-references • regression testing environment (data generation, execution, and analysis of results) • information system linking documentation and code.
Maintenance methods:
<ul style="list-style-type: none"> • rigorous impact analysis, planning, and scheduling procedures • systematic and disciplined update procedures for user and system documentation • user communication channels and procedures

Figure 4. The first level of abstraction of taxonomies of relevant maintenance methods and tools (see [BC91])

3.3 Process taxonomies

Our experience has shown that most of the information needed to carry out our assessment process is contained in the organizational model (built in step 1). The process model built in steps 2 and 3 is also necessary, but it can be fairly simple and straightforward. The process model we built for our case study (described in section 4.2) is simply a breakdown of the maintenance process into phases, with activities and relevant documents identified for each phase. Thus, the supporting technologies needed for the process modeling step are simply a taxonomy of maintenance documents and a taxonomy of generic activities. These taxonomies are shown in Figures 5 and 6, respectively.

Product-related:
<ul style="list-style-type: none">• software requirements specifications• software design specifications• software product specifications
Process-related:
<ul style="list-style-type: none">• test plans• configuration management plan• quality assurance plan• software development plan
Support-related:
<ul style="list-style-type: none">• software user's manual• computer systems operator's manual• software maintenance manual• firmware support manual

Figure 5. A generic taxonomy of maintenance documentation (see [BC91]).

The taxonomy of generic maintenance activities is shown in Figure 6. All these activities usually contain an overhead of communication (meeting and release document writing) with owners, users, management hierarchy and other maintainers, which should be estimated. This is possible through data collection or by interviewing maintainers.

Acronym	Activity
DET	Determination of the need for a change
SUB	Submission of change request
UND	Understanding requirements of changes: localization, change design prototype
IA	Impact analysis
CBA	Cost/benefit analysis
AR	Approval/rejection/priority, assignment of change request
SC	Scheduling/planning of task
CD	Change design
CC	Code changes
UT	Unit testing of modified parts, i.e., has the change been implemented?
IC	Unit Inspection, Certification, i.e., has the change been implemented properly and according to standards?
IT	Integration testing, i.e., does the changed part interface correctly with the reconfigured system?
RT	Regression testing, i.e., does the change have any unwanted side effects?
AT	Acceptance testing, i.e., does the new release fulfill the system requirements?
USD	Update system and user documentation
SA	Checking conformance to standards; quality assurance procedures
IS	Installation
PIR	Post-installation review of changes
EDU	Education/training regarding the application domain/system

Figure 6. Taxonomy of generic maintenance activities (see [BC91])

Error origin: when did the misunderstanding occur?
<ul style="list-style-type: none"> • Change requirements analysis • Change localization analysis • Change design analysis • Coding
Error domain: what caused it?
<ul style="list-style-type: none"> • Lack of application domain knowledge: <i>operational constraints (user interface, performance), mathematical model</i> • Lack of system design or implementation knowledge: <i>data structure or process dependencies, performance or memory constraints, module interface inconsistency</i> • Ambiguous or incomplete requirements • Language misunderstanding <semantic, syntax> • Schedule pressure • Existing uncovered fault • Oversight.

Figure 7. Taxonomy of human errors.

3.4 Causal analysis technologies

The causal analysis part of the assessment method requires several taxonomies which are used to categorize the problems found. The first taxonomy required is one of human errors which lead to maintenance problems. This taxonomy is shown in Figure 7. Another substep in causal analysis is to categorize the findings according to a taxonomy of common maintenance process and organization flaws. The taxonomy we used, another piece of supporting technology, is shown in Figure 8. As a guide for conducting interviews and studying release documents, an outline of the information that should be collected for each software change is provided in Figure 9.

Organizational flaws:
<ul style="list-style-type: none">• communication: interface problems, information flow "bottlenecks" in the communication between the maintainers and the<ul style="list-style-type: none">• users• management hierarchy• quality assurance (QA) team• configuration management team• roles:<ul style="list-style-type: none">• prerogatives and responsibilities are not fully defined or explicit• incompatible responsibilities, e.g., development and QA• process conformance: no effective structure for enforcing standards and processes
Maintenance methodological flaws
<ul style="list-style-type: none">• Inadequate change selection and priority assignment process• Inaccurate methodology for planning of effort, schedule, personnel• Inaccurate methodology for impact analysis• Incomplete, ambiguous protocols for transfer, preservation and maintenance of system knowledge• Incomplete, ambiguous definitions of change requirements• Lack of rigor in configuration (versions, variations) management and control• Undefined / unclear regression testing success criteria.
Resource shortages
<ul style="list-style-type: none">• Lack of financial resources allocated, e.g., necessary for preventive maintenance, unexpected problems unforeseen during impact analysis.• Lack of tools providing technical support (see previous tool taxonomy)• Lack of tools providing management support (i.e., impact analysis, planning)
Low quality product(s)
<ul style="list-style-type: none">• Loosely defined system requirements• Poor quality design, code of maintained system• Poor quality system documentation• Poor quality user documentation
Personnel-related issues
<ul style="list-style-type: none">• Lack of experience and/or training with respect to the application domain• Lack of experience and/or training with respect to the system requirements (hardware, performance) and design• Lack of experience and/or training with respect to the users' operational needs and constraints

Figure 8. Taxonomy of maintenance process flaws

1 Description of the change	<ul style="list-style-type: none"> How long has the person been working on the system? How long has the person been working in this application domain?
1.1 Localization	
<ul style="list-style-type: none"> subsystem(s) affected module(s) affected inputs/outputs affected 	
1.2 Size	
<ul style="list-style-type: none"> LOCs deleted, changed, added Modules examined, deleted, changed, added 	2.3 Did the change generate a change in any document? Which document(s)?
1.3 Type of change	3 Description of the problem
<ul style="list-style-type: none"> Preventive changes: improvement of clarity, maintainability or documentation. Enhancement changes: add new functions, optimization of space/time/accuracy Adaptive changes: adapt system to change of hardware and/or platform Corrective changes: corrections of development errors. 	3.1 Were some errors committed?
2 Description of the change process	<ul style="list-style-type: none"> Description of the errors (see taxonomies in Figure 7) Perceived cause of the errors: maintenance process flaw(s) (see Figure 8)
2.1 effort, elapsed time	3.2 Difficulty
2.2 maintainer's expertise and experience	<ul style="list-style-type: none"> What made the change difficult? What was the most difficult activity associated with the change?
	3.3 How much effort was wasted (if any) as a result of maintenance process flaws?
	3.4 What could have been done to avoid some of the difficulty or errors (if any)?

Figure 9 Guide to data collection in Step 5.

4. Case Study

In the subsections below, the case study we conducted is described. The individual steps of the maintenance process assessment method, as they were implemented in the case study, are described in detail. For each step, a set of substeps and/or guidelines is presented which facilitates the understanding and implementation of the step. In addition, those steps that are iterative in nature include an explanation of how they fit into the general qualitative analysis process shown in Figure 1.

This case study was performed with the team maintaining GTDS (Goddard Trajectory Determination System), a 26 year old, 250 KLOC, FORTRAN orbit determination system. It is public domain software and, as a consequence, has a very large group of users all over the world. Usually, 1 or 2 releases are produced every year in addition to mission specific versions that do not go into configuration management right away (but are integrated later into a new version by going through the standard release process). Like most maintained software systems, very few of the original developers are still present in the organization and turnover still remains a crucial issue in this environment.

GTDS has been maintained by the Flight Dynamics Division (FDD) of the NASA Goddard Space Flight Center for the last 26 years and is still used daily for most operating satellites. Our case study takes place in the framework of the NASA Software Engineering Laboratory (NASA-SEL), an organization aimed at improving FDD software development processes based on measurement and empirical analysis. Recently, responding to the growing cost of software maintenance, the NASA-SEL has initiated a program aimed at characterizing, evaluating and improving its maintenance processes. The maintenance process assessment methodology presented in this paper was created as part of that effort.

4.1 Modeling the Organization

Step 1 Identify the organizational entities with which the maintenance team interacts and the organizational structure in which maintainers operate.

The output of this step is a model which represents the organizational context of the maintenance process. Building this model is a very important step in the analysis process. Gathering organizational information and constructing the model is critical to understanding the work environment. This understanding makes it possible to accurately analyze flaws in the environment later in the analysis process. This model is also useful in checking the consistency and completeness of the maintenance process model constructed in Steps 2 and 3. For example, the organizational model allows us to determine whether or not all roles implied by the process (based on its constituent activities) are officially assigned to organizational actors.

Like many steps in the assessment process, this first step is iterative. In order to illustrate this, we map this step back into the qualitative analysis process shown in Figure 1. Executing this step usually corresponds to a set of iterations of the qualitative analysis process. The input into the process consists of (structured) interviews, organization charts, maintenance standards definition documents, and samples of release documents. These elements comprise the *Observational Database* (ODB). The organization model, which includes roles, agents, teams, information flow, etc., is the resulting characterization model that goes into the *Interpretative Knowledge Base* (IKB). The *validation* procedure helps verify the correctness of the organization model. Questions asked during validation include the following:

- Are all the standard documents and artifacts included in the modeled information flow?
- Do we know who produces, validates, and certifies the standard documents and artifacts?
- Are all the people referenced in the release documents a part of the organization model?

The answers to these questions motivate the collection of more material for the ODB. The process iterates with updates and modifications to the organization model (IKB).

We have defined three major subtasks which comprise this first step. The focus of each step is a particular type of information that should be included in the resulting organization model.

- 1.1 Identify distinct organizational entities, i.e., what are the distinct roles, teams, and working groups involved in the maintenance project?
- 1.2 Characterize various types of dependencies between entities, e.g., information flows: the types and amounts of information, particularly documents, flowing between organizational entities.
- 1.3 Characterize the working environment of each entity. In particular, knowledge of the tools and methods (or lack thereof) available to maintainers is useful in identifying and understanding potential sources of problems.

For the case study, we built a model of the entire NASA-FDD maintenance organization. A simplified version of this model is shown as an A-D model (see section 3.1) in Figure 10. In the sections below, we present the experience gained building and using this model. First, we present the procedures we used for gathering the information we needed to begin building the model. Then we present the details of the model itself.

4.1.1. Acquisition process

Any modeling effort requires that a great deal of information be collected from the environment being modeled. Building an AD model requires collecting information about many people in the environment, the details of their jobs and assignments, whom they depend on to complete their tasks and reach their goals, etc. Our experience has shown that it is useful to follow a defined process for gathering this information, which we will call an *acquisition process*. The acquisition process which we followed, with modifications motivated by our experience, is briefly presented in this section. The steps are as follows:

- A1: First, we determine the official, (usually) hierarchical structure of the organization. Normally this information can be found in official organization charts. This gives us the set of positions and the basic reporting hierarchy.
- A2: We determine the roles covered by the positions by interviewing the people in each position, and then, to check for consistency, their supervisors and subordinates. Process

descriptions, if available, often contain some of this information. However, when using process descriptions, the modeler must check carefully for process conformance.

- A3: In this step, we focus on the goal, resource, and task dependencies that exist along the vertical links in the reporting hierarchy. To do this, we interview members of different departments or teams, as well as the supervisors of those teams. Also, direct observation of supervisors, called "shadowing", can be useful in determining exactly what is requested of, and provided by supervisors for their subordinates.
- A4: Next we focus on resource (usually informational) and goal dependencies between members of the same team. Direct observation (through shadowing or observation of meetings) is also useful here. Interviews and process documents can also be used to identify dependencies.
- A5: Finally, we determine the informational and goal dependencies between different teams. These are often harder to identify, as they are not always explicit. Direct observation is especially important here, as often actors do not recognize their own subtle dependencies on other teams. It is also very important in this step to carefully check for enforcement, assurance, and insurance mechanisms, since dependers and dependees work in different parts of the management hierarchy, given that they belong to different teams.

4.1.2. The Organization Model

The organizational model in Figure 10 is very complex despite important simplifications (e.g., agents and roles are not included). This shows how intricate the network of dependencies in a large software maintenance organization can be.

The model is by necessity incomplete. We have focused on those positions and activities which contribute to the maintenance process only. So there are many other actors in the NASA-FDD organization which do not appear in the A-D graph. As well, we have aggregated some of the positions where appropriate. For example, Maintenance Management includes a large number of separate actors, but for the purposes of our analysis, they can be treated as an aggregate. Below are listed the positions shown in the figure, and a short explanation of their specific roles:

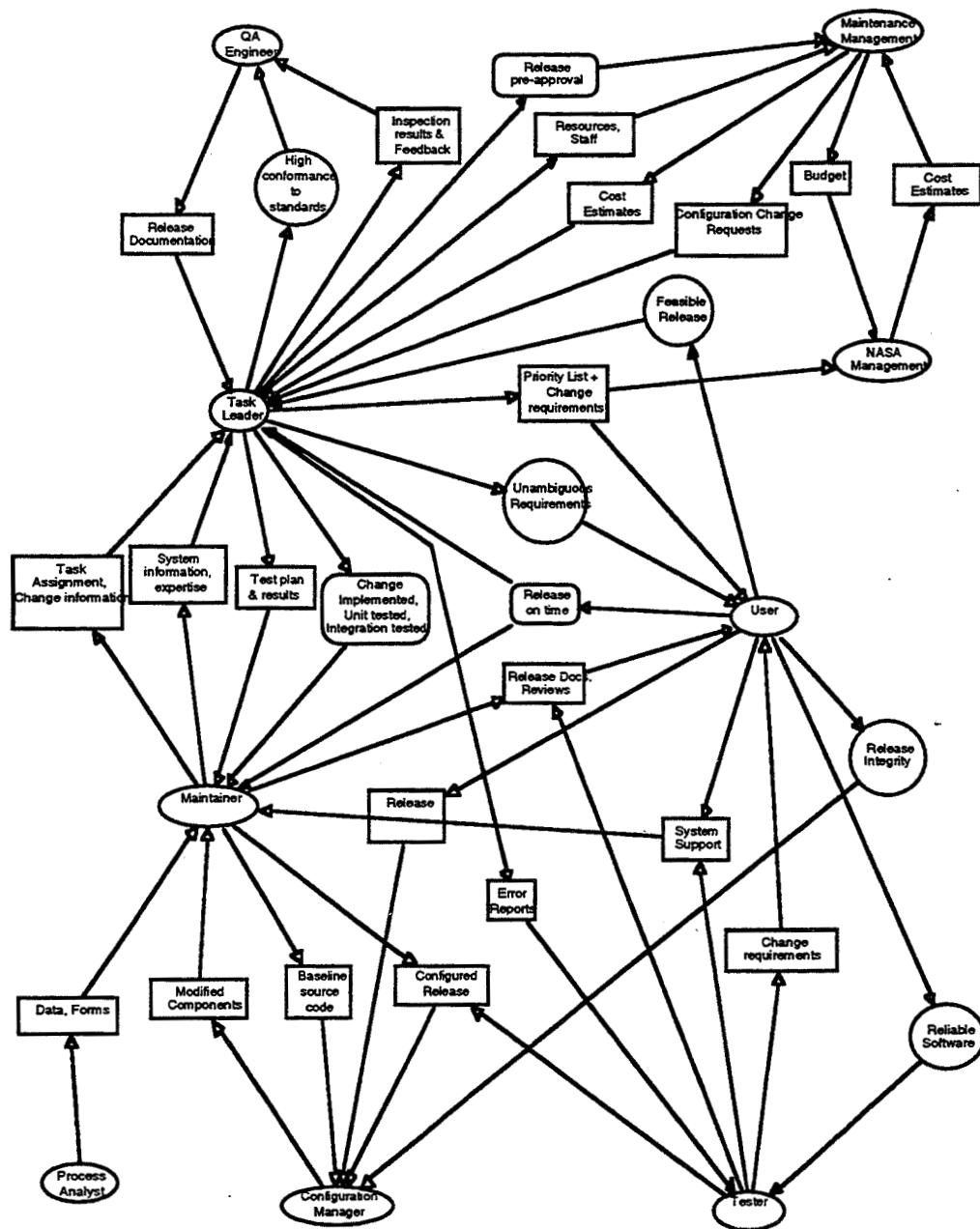


Figure 10 AD Model of a Maintenance Organization.

- *Testers* present acceptance test plans, perform acceptance test and provide change requests to the maintainers when necessary.
- *Users* suggest, control and approve performed changes.
- *QA Engineer* controls maintainers' work (e.g., conformance to standards), attends release meetings, and audits delivery packages.
- *Configuration Manager* integrates updates into the system, coordinates the production and release of versions of the system, and provides tracking of change requests.

- *Maintenance management* grants preliminary approvals of maintenance change requests and release definitions.
- *Maintainers*: analyze changes, make recommendations, perform changes, perform unit and change validation testing after linking the modified units to the existing system, perform validation and regression testing after the system is recompiled by the *Configuration Manager*.
- *Process Analyst* collects and analyzes data from all projects and packages data to be reused.
- *NASA Management* is officially responsible for selecting software changes, gives official authorizations, and provides the budget.

The resulting organizational model was validated through use, within the context of the maintenance assessment methodology. The modeling of the maintenance process, the release documents, and the causal analysis of maintenance problems allowed us to check the model for consistency and completeness.

We also collected data on the maintenance tools and methods which were available and in use. This data collection effort, along with input from the literature, contributed to the creation of the taxonomy of tools and methods in Figure 4. This information turned out not to be relevant in the causal analysis step in this study, so we will not take the space to present it here. However, we believe that information about tools and methods is very important to collect and understand in order to consider all possible sources of maintenance problems.

4.2 Modeling the Process

Step 2 Identify the phases involved in the creation of a new system release.

Step 3 Identify the generic activities involved in each phase.

Phases and activities are defined and differentiated in the following way:

- Phases are ordered tasks with clearly defined milestones and deliverables going through a review and formal approval process.
- Activities are tasks which cannot be a priori ordered within a phase and do not produce deliverables going through a formal approval process although they can be reviewed (e.g., peer reviews).

- Phases contain activities but activities may belong to several phases, e.g., coding may take place during requirement analysis (e.g., prototyping) and, of course, during implementation.

Steps 2 and 3, together, result in the construction of a process model for a maintenance environment. Both project phases and activities may be defined at several levels of decomposition depending on their complexity. It is important to note that the goal here is to better understand the particular release process of the studied environment and not to enact and/or support such a process. We have separated Step 2 from Step 3 because we have found it useful in practice, based on the differences presented above, to separate the characterization of the process into two levels of abstraction. For example, looking at the distribution of activities across phases is often enlightening. The appropriate granularity of a process model is still, from a general perspective, an open issue but can usually be addressed in practice.

Like Step 1, these two steps together are iterative, and thus we can map them back into the qualitative analysis process shown in Figure 1. The material in the *Observational Database* (ODB), with which we begin the process, consists of (in decreasing order of importance) maintenance standards definition documents, interviews, release documents, and the organization model from Step 1. The resulting output is the process model which becomes part of the *Interpretative Knowledge Base* (IKB). The *validation* procedure helps verify the correctness of the process model. Validation questions include:

- Are all the people in the process model a part of the organization model?
- Do the documents and artifacts included in the process model match those of the information flow of the organization model?
- Is the mapping between activities and phases complete, i.e., an exhaustive set of activities, a complete mapping?
- Are a priori relevant types of activities (e.g., defined in Figure 6) missing from the process model?

As before, these questions motivate the collection of more data to be collected in the ODB, which in turn modifies the process model and possibly the organization model (IKB), thus continuing the iterative qualitative analysis process. It is also important to continuously verify that the taxonomies of maintenance tools, methods, and activities are adequate, i.e., that the classes are unambiguous, disjoint and exhaustive.

We have identified the following subtasks for Step 2 (identifying phases):

- 2.1 Identify the phases as defined in the environment studied. At this stage, it is important to perform a bottom-up analysis and avoid mapping (consciously or not) an *a priori* external/generic maintenance process model and terminology.
- 2.2 Each artifact (e.g., document, source code) which is input or output of each phase has to be identified. The taxonomy in Figure 5 is used for this purpose.
- 2.3 The personnel in charge of producing and validating the output artifacts of each phase must be identified and located in the organization model defined in Step 1.

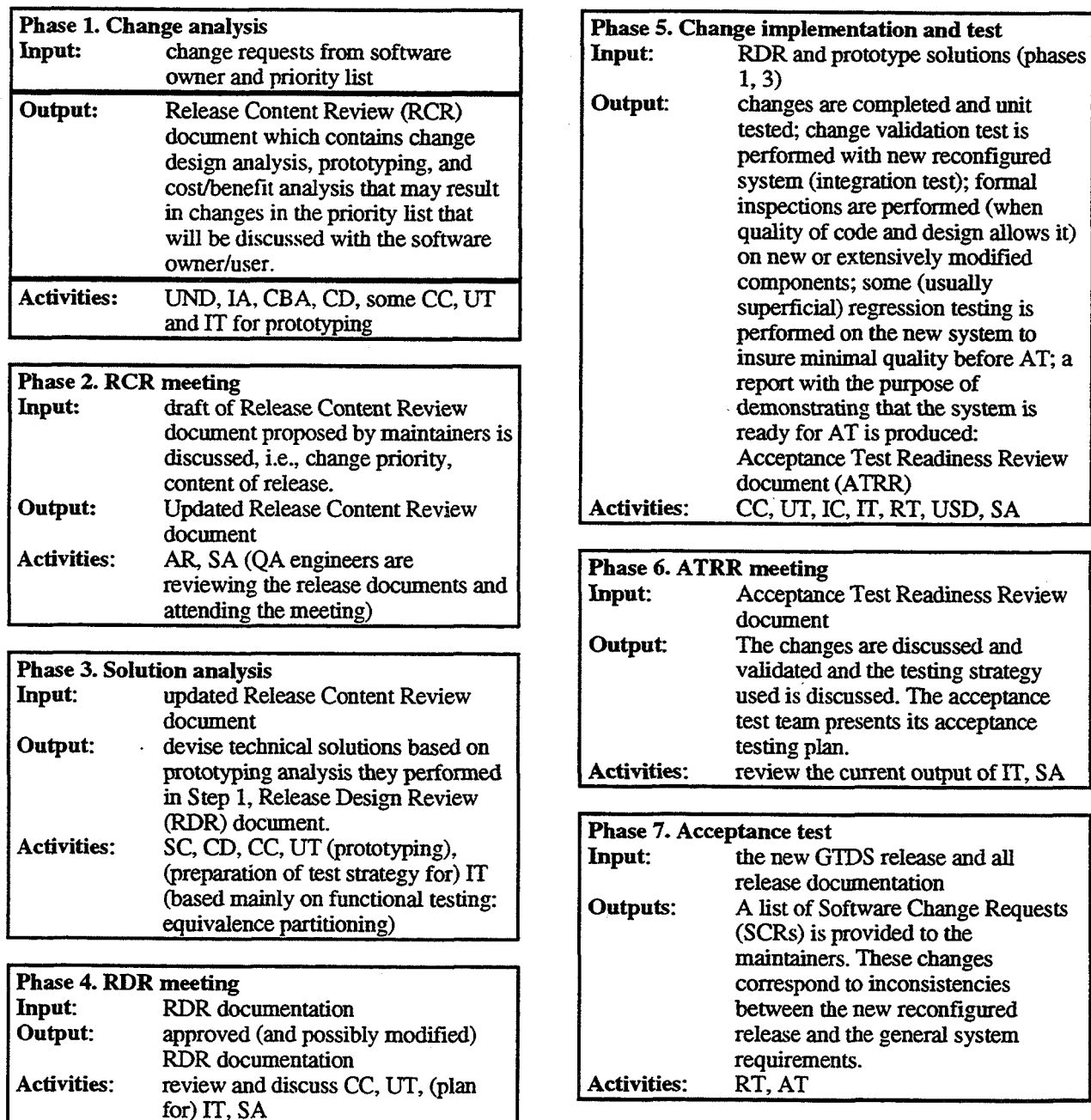


Figure 11. Overview of the Process Model

The process shown in Figure 11 represents our partial understanding of the working process for a release of GTDS and the mapping into standard generic activities (using the taxonomy in Figure 6). This combines the information gained from Steps 2 and 3 of the assessment process. Activity acronyms are used as defined in Figure 6. In this case, each phase milestone in a release is represented by the discussion, approval and distribution of a specific release document (which are defined in the taxonomy shown in Figure 5).

4.3 Selecting Releases for Analysis

Step 4 Select one or several past releases for analysis.

We need to select releases on which we can analyze problems as they are occurring and thereby better understand process and organization flaws. However, because of time constraints, it is sometimes more practical to work on past releases. We present below a set of guidelines for selecting them:

- Recent releases are preferable since maintenance processes and organizational structure might have changed and this would make analyses based on old releases somewhat irrelevant.
- Some releases may contain more complete documentation than others. Documentation has a very important role in detecting problems and cross-checking the information provided by the maintainers.
- The technical leader(s) of a release may have left the company whereas another release's technical leader may still be contacted. This is a crucial element since, as we will see, the causal analysis process will involve project technical leader(s) and, depending on his/her/their level of control and knowledge, possibly the maintainers themselves.

The release selected for analysis in the case study was quite recent, most of the documentation identified in Step 2 was available, and most importantly, the technical leader of the release was available for additional insights and information.

4.4 Causal Analysis

Step 5 Analysis of the problems that occurred while implementing the software changes in the selected releases.

For each software change (i.e., error correction, enhancement, adaptation) in the selected release(s), information should be gathered about the difficulty of the change and any problems that arose during the implementation of the change. This information can be acquired by interviewing the maintainers and/or technical leaders and by reading the related documentation (e.g., release intermediate and final deliverables, error report forms from system and acceptance test)

This step, like several of the previous steps, is iterative, and can thus be mapped into the qualitative analysis process shown in Figure 1. This step usually corresponds to a set of iterations of the qualitative analysis process. The *input* to causal analysis (the contents of the *Observational Database* (ODB)) consists of the results of interviews, change request forms, release deliverables, the organization model (from Step 1), the process model (from Step 2 and 3), and maintenance standards definition documents. The *output* of each iteration is the actual results of the causal analysis, described in the next section. These results constitute the *Interpretative Knowledge Base* (IKB). The *validation* procedure helps verify that the taxonomies of errors and maintenance process flaws are adequate, i.e., unambiguous, disjoint and exhaustive classes. This is checked against actual change data and validated during interviews with maintainers.

The following subtasks of Step 5 define the types of information that should, to the extent possible, be gathered and synthesized during causal analysis. For each software change implemented:

- 5.1. Determine the difficulty or error-proneness of the change.
- 5.2. Determine whether and how the change difficulty could have been alleviated or the error(s) resulting from the change avoided.
- 5.3. Evaluate the size of the change (e.g., # components, LOCs changed, added, removed).
- 5.4. Assess discrepancies between initial and intermediate planning and actual effort / time.
- 5.5. Determine the human flaw(s) (if any) that originated the error(s) or increased the difficulty related to the change (using the taxonomy shown in Figure 7).
- 5.6. Determine the maintenance process flaws that led to the identified human errors (if any), using the taxonomy of maintenance process flaws proposed in Figure 8.
- 5.7. Try to quantify the wasted effort and/or delay generated by the maintenance process flaws (if any).

The knowledge and understanding acquired through steps 1-3 of the assessment process are necessary in order to understand, interpret and formalize the information in substeps 5.2, 5.5 and 5.6. The guide in Figure 9 facilitates subtasks 5.1-5.4.

Step 5 involved a causal analysis of the problems observed during maintenance and acceptance test of the release studied. These problems were linked back to a precise set of issues belonging to taxonomies presented in Figures 7 and 8. Figure 12 summarizes Step 5 as instantiated for this case study. This step required extensive collaboration from the GTDS maintenance task leader, as well as examination of the documents generated during the release process. A questionnaire was also used to gather additional information regarding changes that were part of the release studied. The questionnaire used the taxonomies presented for Step 5. Changes that generated error correction requests from the acceptance testing team were analyzed in particular detail.

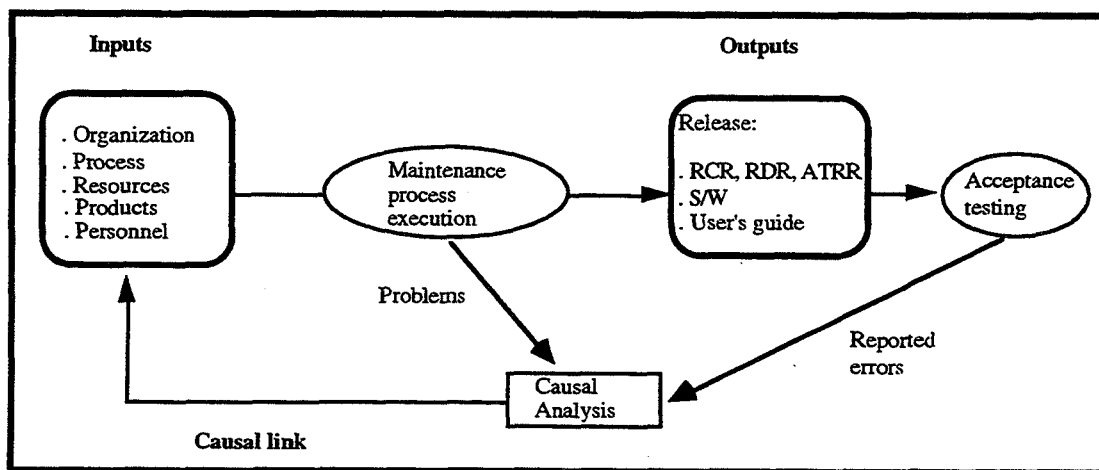


Figure 12: Causal Analysis Process

In order to illustrate Step 5, we provide below an example of causal analysis for *one* of the changes in the selected release (change 642). Implementation of this change resulted in 11 errors that were found by the acceptance test team, 8 of which had to be corrected before final delivery could be made. In addition, a substantial amount of rework was necessary. Typically, changes do not generate so many subsequent errors, but the flaws that were present in this change are representative of maintenance problems in GTDS. In the following paragraphs, we discuss only *two* of the errors generated by the change studied (errors A1044 and A1062).

Change 642 Description: Initially, users requested an enhancement to existing GTDS capabilities. The enhancement involved vector computations performed over a given time span. This enhancement was considered quite significant by the maintainers, but users failed to supply adequate requirements and did not attend the RCR meeting. Users

did not report their dissatisfaction with the design until ATRR meeting time, at which time requirements were rewritten and maintainers had to perform rework on their implementation. This change took a total of 3 months to implement, of which at least 1 month was attributed to rework.

Maintenance process flaw(s):

Organizational: a lack of clear definitions of the prerogatives/duties of users with respect to release document reviews and meetings (roles), and a lack of enforcement of the release procedure (process conformance); methodological: incomplete, ambiguous definitions of change requirements.

Errors caused by change 642

The implementation of the change itself resulted in an error (A1044) found at the acceptance test phase. When the correction to A1044 was tested, an error (A1062) was found that could be traced back to both 642 and A1044.

A1044

Description: Vector computations at the endpoints of the time span were not handled correctly.

But in the requirements it was not clear whether the endpoints should be considered when implementing the solution.

Error origin: change requirement analysis

Error domain: ambiguous and incomplete requirements

Maintenance process flaw(s):

Organizational: communication between users and maintainers, due in part to a lack of defined standards for writing change requirements; methodological: incomplete, ambiguous definitions of change requirements.

A1062

Description: One of the system modules in which the enhancement change was implemented has two processing modes for data. These two modes are listed in the user manual. When run in one of the two possible processing modes, the enhancement generated a set of errors, which were put under the heading A1062. At the phase these errors were found, the enhancement had already successfully passed the tests for the other processing mode. The maintainer should have designed a solution to handle both modes correctly.

Error origin: change design analysis.

Error domain: lack of application domain knowledge.

Maintenance process flaw(s):

Personnel-related: lack of experience and/or training with respect to the application domain.

4.5 Synthesis

Step 6 Establish the frequency and consequences of problems due to flaws in the organizational structure and the maintenance process by analyzing the information gathered in Step 5.

Based on the results from Step 5, further complementary investigations (e.g., measurement-based), related to specific issues that have not been fully resolved by the qualitative analysis process, should be identified. Moreover, a first set of suggestions for maintenance process improvement should be devised.

The lessons learned are classified according to the taxonomy of maintenance flaws defined in Figure 8. By performing an overall analysis of the change causal analysis results (Step 6), we abstracted a set of issues detailed in the following sections.

4.5.1. Organization

- There is a large communication cost overhead between maintainers and users, e.g., release standard documentation, meetings, and management forms. In an effort to improve the communication between all the participants of the maintenance process, non-technical, communication-oriented activities have been emphasized. At first glance, this seems to represent about 40% (rough approximation) of the maintenance effort. This figure seems excessive, especially when considering the apparent communication problems (next paragraph).
- Despite the number of release meetings and documents, disagreements and misunderstandings seem to disturb the maintenance process until late in the release cycle. For example, design issues that should be settled at the end of the RDR meeting keep emerging until acceptance testing is completed.

As a result, it seems that the administrative process and organization scheme should be investigated in order to optimize communication and sign-off procedures, especially between users and maintainers.

4.5.2. Process

- The tools and methodologies used have been developed by maintainers themselves and do not belong to a standard package provided by the organization. Some ad hoc technology transfer seems to take place in order to compensate for the lack of a global, commonly agreed upon strategy.
- The task leader has been involved in the maintenance of GTDS for a number of years. His expertise seems to compensate for the lack of system documentation. He is also in charge of the training of new personnel (some of the easy changes are used as an opportunity for training). Thus, the process relies heavily on the expertise of one or two persons.
- The fact that no historical database of changes exists makes some changes very difficult. Maintainers very often do not understand the semantics of a piece of code added in a previous correction. This seems to be partly due to emergency patching for a mission which was not controlled and cleaned up afterwards (this has recently been addressed), a high turnover of personnel, and a lack of written requirements with respect to performance, precision and platform configuration constraints.
- For many of the complex changes, requirements are often ambiguous and incomplete, from a maintainer's perspective. As a consequence, requirements are often unstable until very late in the release process. While prototyping might be necessary for some of them, it is not recognized as such by the users and maintainers. Moreover, there is no well defined standard for expressing change requirements in a style suitable for both maintainers and users.

4.5.3. Products

- System documentation other than the user's guide is not fully maintained and not trusted by maintainers. Source code is currently the only reliable source of information used by maintainers.
- GTDS has a large number of users. As a consequence, the requirements of this system are varied with respect to the hardware configurations on which the system must be able to run, the performance and precision needs, etc. However, no requirement analysis document is available and maintained in order to help the maintainers devise optimal change solutions.
- Because of budget constraints, there is no document reliably defining the hardware and precision requirements of the system. Considering the large number of users and

platforms on which the system runs, and the rapid evolution of users' needs, this would appear necessary in order to avoid confusion while implementing changes.

4.5.4. People

- There is a lack of understanding of operational needs and constraints by maintainers. Release meetings were supposed to address such issues but they seem to be inadequate in their current form.
- Users are mainly driven by short term objectives which are aimed at satisfying particular mission requirements. As a consequence, there is a very limited long term strategy and budget for preventive maintenance. Moreover, the long term evolution of the system is not driven by a well defined strategy and maintenance priorities are not clearly identified.

4.5.5. General Recommendations

As a general set of recommendations and based on the analysis presented in this paper, we suggested the following set of actions to the GTDS maintenance project:

- A standard (that may simply contain guidelines and checklists) should be set up for defining and documenting change requirements. Both users and maintainers should give their input with respect to the content of this standard since it is intended to help them communicate with each other.
- The conformance to the defined release process should be improved, e.g., through team building and training. In other words, the release documents and meetings should more effectively play their specified role in the process, e.g., the RDR meeting should settle all design disagreements and inconsistencies.
- Those parts of the system that are highly convoluted as a result of numerous modifications should be redesigned and documented for more productive and reliable maintenance. Technical task leaders should be able to point out the sensitive system units.

5. Quantitative Analysis

The use of quantitative data is critical to the useful analysis of development processes and organizations. Quantitative information is needed to effectively compare alternatives and to make decisions. However, as mentioned earlier, quantitative endeavors can be expensive and take time

to initiate. For this reason, qualitative approaches like the one presented in this paper are necessary to obtain meaningful insights in a reasonable period of time. But qualitative analysis must be taken further to provide a basis for action. And qualitative approaches are best when they are designed to incrementally incorporate quantitative results as they become available.

There is a need to clearly define the quantitative information that needs to be collected and its relationship to organization and process models. This careful definition of data entities must take place when a quantitative measurement program is being planned and designed. The data entities themselves must be identified, along with their relevant attributes, and the relationships between entities must be defined. Entity-Relationship-Attribute (ERA) models are often used for this purpose. Such a model helps clarify data collection and analysis issues, as well as to define how the data will be stored. The partial E-R model shown in Figure 13 (we've omitted the attributes for this discussion) is a generic template that describes how quantitative information about process and organization could be stored together. This E-R model does not intend to be complete but can be refined to fit the needs of the measurement program being designed. For example, phases could be decomposed through a reflexive "Is part of" relationship between Process Phases entities. The attributes that will characterize the entities will depend on the goals of the data collection, the resources available, and specifics of the studied environment and process. Visualization, enactment, and analysis tools can be built upon such a database and provide a consistent process-centered environment for improvement.

AD models are particularly well suited to incorporating data, although there is not an explicit facility for this in the modeling methodology. One way to perform such analysis is to associate attributes with the various AD entities (positions, roles, dependencies, etc.). The attributes could be used to hold the quantitative information. Then analysis tools can be used to analyze the AD graph, by making calculations, based on the data, according to the structure represented in the graph.

In building the E-R model in Figure 13, we began with the entities already present in A-D models, then added others we felt were relevant for the quantitative analysis of maintenance processes and organizations. One entity that we have added in Figure 13 is the Qualification entity. An agent "has" one or more qualifications, e.g., maintaining ground satellite software systems. Moreover, based on experience, it may be determined that some role "requires" specific qualifications, e.g., experience with Ada. Comparison of the required qualifications and the actual organizational set-up appears useful for identifying high-risk organizational patterns.

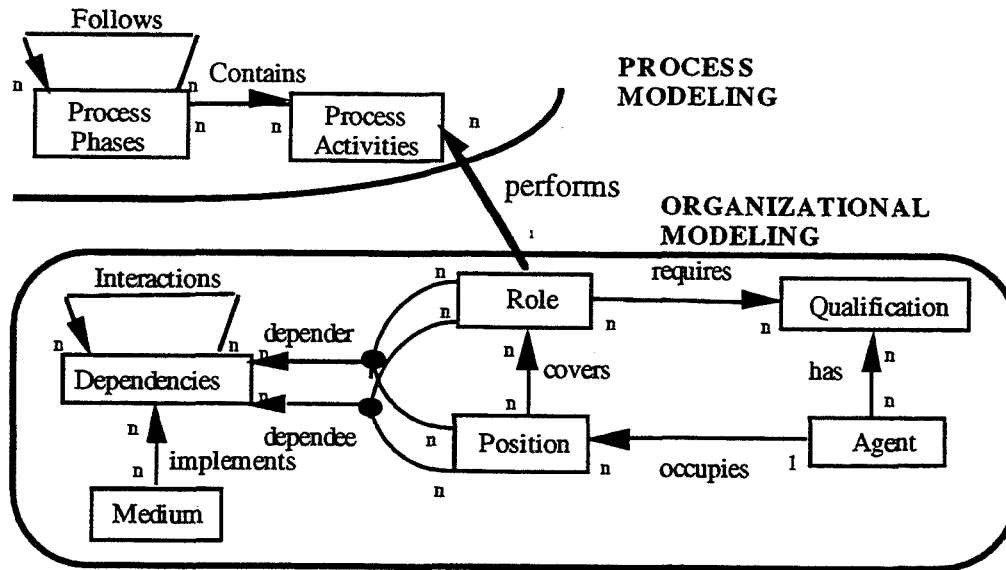


Figure 13: ER model for quantitative analysis using AD graphs

We have retained the agent/role/position decomposition of an actor defined by the A-D modeling formalism, which we found very useful. The E-R model also shows "dependers" and "dependees" as ternary relationships. This reflects the fact that a dependers or dependees of a dependency can be either a role or a position. A role may be functionally dependent on another role in order to perform a given process activity. Interdependent positions are usually so because of the need for authorization or authority. However, we believe that dependencies are not inherent to agents themselves, at least not in our context.

We have also added a new entity, Medium, which is the communication medium used to implement a particular dependency (especially information dependencies). This entity may be used in some types of quantitative analysis. Also, dependencies are related to each via the interaction relationship, which describes the risk management mechanisms (enforcement, assurance, insurance) that are implemented between dependencies.

The E-R model also makes explicit the relationship, and the separation, between process and organization. Analysis of an organization is aided by the isolation of organizational issues (e.g., information flow, distribution of work) from purely process concerns (e.g., task scheduling, concurrency). However, although organization and process raise separate issues, their effects are related. Understanding the relationship between organization and process is crucial to making improvements to either aspect of the environment (requirement **Req4**). For example, the "performs" relationship can link a role to a set of activities, which may be seen as lower-level

roles. The entity Process Activity is itself related to other entities in the process model that are not specified in Figure 13, e.g., process artifacts.

One type of quantitative analysis is information flow analysis. Information dependencies (one type of resource dependency) can have attached to them attributes such as frequency and amount of information. Each information dependency is also related to the different communication media that it uses to pass information, e.g. phone, email, formal and informal documents, formal and informal meetings. The many-to-many relationships between dependencies and their media can also have attributes (e.g., effort). Such attributes are captured by defining metrics and collecting the appropriate data. An example of such an attribute is the computation, for each information dependency, of the product of the dependency frequency, the amount of information, and the effort associated with the medium related to the dependency. This product gives a quantitative assessment of the effort expended to satisfy the information dependency. Summing these values for each pair of actors in the AD graph shows how much effort the pair expends in passing information to each other. This information can be used to support such management decisions as how to fill different positions, how to locate these people, and what communication media to make available. This is just one example of how A-D models can be used along with measurement to provide quantitative results for the purposes of decision making. Without quantitative analysis, these decisions are subject to guesswork, trial and error, and the personal expertise of the manager. For more on metrics for organizational information flow, see [S94].

There are several possible applications of quantitative analysis in relation to the actor/position/role decomposition. For example, during the course of our study, we noticed that many differences between projects were reflected in variations in the breakdown of positions into roles. In other words, the people filling the same positions in different projects divided their effort differently among their various roles. These variations were usually symptomatic of differences in management strategy and leadership style. Data needs to be collected to capture the important variations in effort breakdown across organizations and projects. This data must then be attached to entities in the AD model so that it can be used to analyze variations in job structure. For example, suppose that we wanted to find out which projects require a manager with technical expertise. If we have quantitative data available on the effort breakdown of the different managers, then we can easily see which managers spend a high proportion of their time on technical activities. This information can be used in choosing people to fill different management positions.

Another example of the many possibilities for analysis of the role/position/agent structure of actors is qualification analysis where required and actual qualifications are compared for roles

and positions. Understanding the sharing of tasks and responsibilities is another area in which quantitative analysis could be useful. All of these involve the evaluation of quantitative attributes attached to roles, positions, agents, and the links (occupies, contains, performs) between them.

6. Conclusion

Characterizing and understanding software maintenance processes and organizations are necessary, if effective management decisions are to be made and if adequate resource allocation is to be provided. Also, in order to plan and efficiently organize a measurement program—a necessary step towards process improvement [BR88]—, we need to better characterize the maintenance environment and its specific problems. The difficulty of performing such a characterization stems from the fact that the people involved in the maintenance process, who have the necessary information and knowledge, cannot perform it because of their inherently partial perspective on the issues and the tight time constraints of their projects. Therefore, a well defined characterization and assessment process, which is cost-effective, objective, and applicable by outsiders, needs to be devised.

In this paper, we have presented such an empirically refined process which has allowed us to gain an in-depth understanding of the maintenance issues involved in a particular project, the GTDS project. We have been able to gather objective information on which we can base management and technical decisions about the maintenance process and organization. Moreover, this process is general enough to be followed in most maintenance organizations.

However, such a qualitative analysis is a priori limited since it does not allow us to quantify precisely the impact of various organizational, technical, and process related factors on maintenance cost and quality. Thus, the planning of the release is sometimes arbitrary, monitoring its progress is extremely difficult, and its evaluation remains subjective.

Hence, there is a need for a data collection program for GTDS and across all the maintenance projects of the organization. In order to reach such an objective, we have to base the design of such a measurement program on the results provided by this study. In addition, we need to model more rigorously the maintenance organization and processes so that precise evaluation criteria can be defined [SB94]. Preliminary results from the current maintenance measurement program can be found in [B+96].

This approach is being used to analyze several other maintenance projects in the NASA-SEL in order to better understand project similarities and differences in this environment. Thus, we will be able to build better models of the various classes of maintenance projects.

7. References

- [B+95] L. Briand, W. L. Melo, C. Seaman, V. Basili. "Characterizing and Assessing a Large-Scale Software Maintenance Organization". ICSE'95, Seattle, WA, 1995.
- [B+96] V. Basili, L. Briand, S. Condon, W. L. Melo, C. Seaman, J. Valett. "Understanding and Predicting the Process of Software Maintenance Releases". ICSE'96, Berlin, Germany, 1996.
- [BC91] K. Bennett, B. Cornelius, M. Munro, D. Robson, "Software Maintenance", *Software Engineering Reference Book*, Chapter 20, Butterworth-Heinemann Ltd, 1991
- [BK94] I.Z. Ben-Shaul and G. Kaiser, "A paradigm for decentralized process modeling and its realization in the OZ environment", ICSE 16, May 1994, Sorrento, Italy.
- [BR88] V. Basili and H. Rombach, "The TAME Project: Towards Improvement-Oriented Software Environments", *IEEE Trans. Software Eng.*, 14 (6), June, 1988.
- [C88] N. Chapin, "The Software Maintenance Life-Cycle", CSM'88, Phoenix, Arizona, 1988.
- [CC93] B.G. Cain and J.O. Coplien, "A Role-Based Empirical Process Modeling Environment", ICSP 2, Berlin, Germany, February 1993.
- [HV92] M. Hariza, J.F. Voidrot, E. Minor, L. Pofelski, and S. Blazy, "Software Maintenance: An analysis of Industrial Needs and Constraints", CSM'92, Orlando, Florida.
- [K91] M.I. Kellner, "Software Process Modeling Support for Management Planning and Control", ICSP 1, Redondo Beach, CA, October 1991.
- [LR93] C.M. Lott and H.D. Rombach, "Measurement-based guidance of software projects using explicit project plans", *Information and Software Technology*, 35:6/7, June, 1993, pp. 407-419.
- [MCC92] "Deva, A Process Modeling Tool", MCC Technical Report, June 1992.

- [R92] G.L. Rein, "Organization Design Viewed as a Group Process Using Coordination Technology", MCC Technical Report CT-039-92, February 1992.
- [RUV92] D. Rombach, B. Ulery and J. Valett, "Toward Full Cycle Control: Adding Maintenance Measurement to the SEL", *Journal of systems and software*, May 1992.
- [S94] C.B. Seaman, "Using the OPT improvement approach in the SQL/DS development environment", in *Proceedings of CASCON '94*, IBM Canada Ltd. Laboratory Centre for Advanced Studies and National Research Council of Canada, Toronto, Canada, October 1994.
- [SB94] C. Seaman and V. Basili, "OPT: An Approach to Organizational and Process Improvement", AAAI 1994 Spring Symposium Series, Stanford University, March 1994.
- [SS92] A. Shelly and E. Sibert, "Qualitative Analysis: A Cyclical Process Assisted by Computer", *Qualitative Analysis*, pp 71-114, Oldenbourg Verlag, Munich, Vienna, 1992
- [YM93] E. Yu and J. Mylopoulos, "An Actor Dependency Model of Organizational Work - with Application to Business Process Reengineering". In *Proc. Conference on Organizational Computing Systems (COOCS 93)*, Milpitas, CA, November 1993.
- [YM94] E. Yu and J. Mylopoulos, "Understanding 'why' in software process modeling, analysis, and design", ICSE 16, Sorrento, Italy, May 1994.

Evolving and Packaging Reading Technologies

57-61
415773

V.R. Basili

Department of Computer Science

and Institute for Advanced Computer Studies

University of Maryland, College Park, MD 20742 U.S.A.

(301)405-2668, (301)405-6707 FAX, basili@cs.umd.edu

360853

Abstract

Reading is a fundamental technology for achieving quality software. This paper provides a motivation for reading as a quality improvement technology, based upon experiences in the Software Engineering Laboratory at NASA Goddard Space Flight Center and shows the evolution of our study of reading via a series of experiments. The experiments range from the early reading vs. testing experiments to various Cleanroom experiments that employed reading to the development of new reading technologies currently under study.

12A

Keywords

Reading scenarios, cleanroom, experiments, inspections, quality improvement paradigm

1. INTRODUCTION

Reading is a fundamental technology for achieving quality software. It is the only analysis technology we can use throughout the entire life cycle of the software development and maintenance processes. And yet, very little attention has been paid to the technologies that underlie the reading of software documents. For example where is "software reading" taught? What technologies have been developed for "software reading"? In fact, what is "software reading"?

During most of our lives, we learned to read before we learned to write. Reading formed a model for writing. This was true from our first learning of a language (reading precedes writing and provides simple models for writing) to our study of the great literature (reading provides us with models of how to write well). Yet, in the software domain, we never learned to read, e.g., we learn to write programs in a programming language, but never learn how to read them. We have not developed reading-based models for writing.

For example, we are not conscious of our audience when we write a requirements document. How will they read it? What is the difference between reading a requirements document and reading a code document? We all know that one reads a novel differently than one reads a text book. We know that we review a technical paper differently than we review a newspaper article. But how do we read a requirements document, how do we read a code document, how do we read a test plan?

Achieving Quality in Software, Proceedings of the Third International Conference, AQUIS'96, January 24-26, 1996, Florence, Italy.

But first let us define some terms so that we understand what we mean by "reading". We differentiate a technique from a method, from a life cycle model. A technique is the most primitive, it is an algorithm, a series of steps producing the desired effect. It requires skill. A method is a management procedure for applying techniques, organized by a set of rules stating how and when to apply and when to stop applying the technique (entry and exit criteria), when the technique is appropriate, and how to evaluate it. We will define a technology as a collection of techniques and methods. A life cycle model is a set of methods that covers the entire life cycle of a software product.

For example, reading by step-wise abstraction [Linger, Mills, and Witt, 1979] is a technique for assessing code. Reading by stepwise abstraction requires the development of personal skills; one gets better with practice. A code inspection is a method, that is defined around a reading technique, which has a well defined set of entry and exit criteria and a set of management supports specifying how and when to use the technique. Reading by stepwise abstraction and code inspections together form a technology. Inspections are embedded in a life cycle model, such as the Cleanroom development approach, which is highly dependent on reading techniques and methods. That is, reading technology is fundamental to a Cleanroom development.

In what follows, we will discuss the evolution and packaging of reading as a technology in the Software Engineering Laboratory (SEL) [Basili, Caldiera, McGarry, Pajerski, Page, Waligora, 1992] via a series of experiments from some early reading vs. testing technique experiments, to various Cleanroom experiments, to the development of new reading techniques currently under study.

In the SEL, we have been working with a set of experimental learning approaches: the Quality Improvement Paradigm, the Goal Question Metric Paradigm, the Experience Factory Organization, and various experimental frameworks to evolve our knowledge and the effectiveness of various life cycle models, methods, techniques, and tools [Basili - 1985, Basili and Weiss - 1984, Basili and Rombach - 1988, Basili - 1989]. We have run a series of experiments at the University of Maryland and at NASA to learn about, evaluate, and evolve reading as a technology.

2. READING STUDIES

Figure 1 provides a characterization of various types of experiments we have run in the SEL. They define different scopes of evaluation representing different levels of confidence in the results. They are characterized by the number of teams replicating each project and the number of different projects analyzed yielding four different experimental treatments: blocked subject-project, replicated project, multi-project variation, and single project case study.

The approaches vary in cost, level of confidence in the results, insights gained, and the balance between quantitative and qualitative research methods. Clearly, an analysis of several replicated projects costs more money but provides a better basis for quantitative analysis and can generate stronger statistical confidence in the conclusions. Unfortunately, since a blocked subject-project experiment is so expensive, the projects studied tend to be small. To increase the size of the projects, keep the costs reasonable, and allow us to better simulate the effects of the treatment variables in a realistic environment, we can study very large single project case studies and even multi-project studies if the right environment can be found. These larger projects tend to involve more qualitative analysis along with some more primitive quantitative analysis.

Because of the desire for statistical confidence in the results, the problems with scale up, and the need to test in a realistic environment, one approach to experimentation is to choose one of the multiple team treatments controlled experiments to demonstrate feasibility (statistical significance) in the small, and then to try a case study or multiproject variation to analyze whether the results scale up in a realistic environment - a major problem in studying the effects

of techniques, methods and life cycle models.

Scopes of Evaluation				
		#Projects		
		One	More than one	
# of Teams	One	Single Project (Case Study)	Multi-Project Variation	
per Project	More than one	Replicated Project	Blocked Subject-Project	

Figure 1: Classes of Studies

2.1 Reading by stepwise abstraction

In order to improve the quality of our software products at NASA, we have studied various approaches. One area of interest was to understand the relationship between reading and testing in our environment. Early experiments showed very little difference between reading and testing [Hetzel - 1972, Myers - 1978]. But reading was simply reading, without a technological base. Thus we attempted to study the differences between various specific technology based approaches. Our goal was to analyze code reading, functional testing and structural testing to evaluate and compare them with respect to their effect on fault detection effectiveness, fault detection cost and classes of faults detected from the viewpoint of quality assurance [Basili, Selby - 1987]. The study was conducted in the SEL, using three different programs: a text formatter, a plotter, and a small database. The programs were seeded with software faults, 9, 6, and 12 faults respectively, and ranged in size from 145 to 365 LOC. The experimental design was a blocked subject-project, using a fractional factorial design. There were 32 subjects.

Specific techniques were used for each of the three approaches studied. Code reading was done by stepwise abstraction, i.e., reading a sequence of statements and abstracting the function they compute and repeating the process until the function of the entire program has been abstracted and can be compared with the specification. Functional testing was performed using boundary value, equivalence partition testing, i.e., dividing the requirements into valid and invalid equivalence classes and making up tests that check the boundaries of the classes. Structural testing: was performed to achieve 100% statement coverage, i.e., making up a set of tests to guarantee that 100% of the statements in the program have been executed.

As a blocked subject-project study, each subject used each technique and tested each program. The results were that code reading found more faults than functional testing, and functional testing found more faults than structural testing. Also, code reading found more faults per unit of time spent than either of the other two techniques.

Other conclusions from the study include the fact that the code readers were better able to assess the actual quality of the code that they read than the testers. And in fact, the structural testers were better able to assess the actual quality of the code they read than the functional testers. That

is, the code readers felt they only found about half the defects (and they were right), where the functional testers felt that had found about all the defects (and they were wrong). Also, after the completion of the study over 90% of the participants thought functional testing worked best. This was a case where their intuition was clearly wrong.

Based upon this study, reading was implemented as part of the SEL development process. However, much to our surprise, reading appeared to have very little effect on reducing defects. This lead us to two possible hypotheses:

Hypothesis 1: People did not read as well as they should have as they believed that testing would make up for their mistakes

To test this first hypothesis, we ran an experiment that showed that if you read and cannot test you do a more effective job of reading than if you read and know you can test. This supported hypothesis 1.

Hypothesis 2: There is a confusion between reading as a technique and the method in which it is embedded, e.g., inspections.

This addresses the concern that we often use a reading method (e.g., inspections or walk-throughs) but do not often have a reading technique (e.g., reading by stepwise abstraction) sufficiently defined within the method. To some extent, this might explain the success of our experiment over the ones by Hetzel and Myers.

Thus we derived the following conclusions from the studies to date:

- Reading using a particular technique is more effective and cost effective than specific testing techniques, i.e., the reading technique is important. However, different approaches may be effective for different types of defects.
- Readers needs to be motivated to read better, i.e., the reading motivation is important.
- We may need to better support the reading process, i.e., the reading technique may be different from the reading method.

2.2 The Cleanroom approach

The Cleanroom approach, as proposed by Harlan Mills [Currit, Dyer, Mills - 1986], seemed to cover a couple of these issues, so we tried a controlled experiment at the University of Maryland to study the effects of the approach.

The goal of this study was to analyze the Cleanroom process in order to evaluate and compare it to a non-Cleanroom process with respect to the effects on the process, product and developers [Selby, Basili, Baker - 1987]. This study was conducted using upper division and graduate students at the University of Maryland. The problem studied was an electronic message system of about 1500 LOC. The experimental design was a replicated project, 15 three-person teams (10 used Cleanroom). They were allowed 3 to 5 test submissions to an independent tester. We collected data on the participants' background, attitudes, on-line activities, and testing results.

The major results were:

- With regard to process, the Cleanroom developers (1) felt they more effectively applied off-line review techniques, while others focused on functional testing, (2) spent less time on-line and used fewer computer resources, and (3) tended to make all their scheduled deliveries
- With regard to the delivered product, the Cleanroom products tended to have the

following static properties: less dense complexity, higher percentage of assignment statements, more global data, more comments, and the following operational properties: the products more completely met the requirements and a higher percentage of test cases succeeded.

- With regard to the effect on the developers, most Cleanroom developers missed program execution, modified their development style, but said they would use the Cleanroom approach again.

2.3 Cleanroom in the SEL

Based upon this success, we decided to try the Cleanroom approach in the SEL [Basili and Green - 1994]. This was the basis for a case study and we used the Quality Improvement Paradigm to set up our learning process. The QIP consists of 6 steps and we define them here relative to the use of Cleanroom:

Characterize: What are the relevant models, baselines and measures? What are the existing processes? What is the standard cost, relative effort for activities, reliability? What are the high risk areas? (Figure 2)

Set goals: What are the expectations, relative to the baselines? What do we hope to learn, gain, e.g., Cleanroom with respect to changing requirements? (Figure 2)

Choose process: How should the Cleanroom process be modified and tailored relative to the environment? E.g., formal methods hard to apply, require skill; may have insufficient data to measure reliability. Allow back-out options for unit testing certain modules.

Execute: Collect and analyze data based upon the goals, making changes to the process in real time.

Analyze: Try to characterize and understand what happened relative to the goals; write lessons learned.

Package: Modify the process for future use.

There were many lessons learned during this first application of the Cleanroom approach in the SEL. However, the most relevant to reading were that the failure rate during test was reduced by 25% and productivity increased by about 30%, mostly due to fact that there was a reduction in the rework effort, i.e., 95% as opposed to 58% of the faults took less than 1 hour to fix. About 50% of code time was spent reading, as opposed to the normal 10%. All code was read by 2 developers. However, even though the developers were taught reading by stepwise abstraction for code reading, only 26% of the faults were found by both readers. This implied to us that the reading technique was not applied as effectively as it should have been, as we expected a more consistent reading result.

During this case study, problems, as specified by the users, were recorded and the process was modified.

Based upon the success of the first Cleanroom case study, we began to define new experiments with the goal of applying the reading technique more effectively. The project leader for first project became process modeler for the next two and we began to generate the evolved version of the SEL Cleanroom Process Model. Thus we moved our experimental paradigm from a case study to a multi-project analysis study. Figure 3 gives an overview of the projects studied to date. A fourth project has just been completed but the results have not yet been analyzed.

Cleanroom has been successful in the SEL. Although there is still some room for improvement in reading and abstracting code formally, a more major concern is the lack of techniques for reading various documents reading, most specifically, requirements documents. This provided our motivation for the continual evolution of reading techniques both inside and outside the Cleanroom life cycle model. Specific emphasis is on improving reading technology

for requirements and design documents.

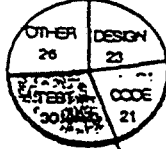
	Sample Measures	Sample Baseline	Sample Expectation
PROCESS	Effort distribution Change profile		Increased design % due to emphasis on peer review process
COST	Productivity Level of rework Impact of spec changes	Historically, 26 DLOC per day	No degradation from current level
RELIABILITY	Error rate Error distribution Error source	Historically, 7 errors per KDLOC	Decreased error rate

Figure 2: Sample Measures, Baselines, and Expectations

Technology Evaluation Steps	Off-line Reading Technology Controlled Experiment	Off-line Cleanroom Controlled Experiment	SEL Cleanroom Case Study 1	SEL Cleanroom Case Study 2	
				project 2A	project 2B
Team Size	32 individual participants	3-person development teams (10 Cleanroom teams, 5 control teams)	3-person development team, 2-person test team	4-person development team, 2-person test team	14-person development team, 4-person test team
Project Size and Application	small (145-365 LOC) sample FORTRAN programs	1500 LOC electronic message system for graduate lab course	40 KDLOC FORTRAN flight dynamics production system	22 KDLOC FORTRAN flight dynamics production system	160 KDLOC FORTRAN flight dynamics production system
Results	reading techniques appear more effective than testing techniques for fault detection	Cleanroom teams use fewer computer resources, satisfy requirements more successfully, make higher percentage of scheduled deliveries	project spends higher percentage of effort in design, uses fewer computer resources, achieves better productivity and reliability than environment baseline	project continues trend in better reliability while maintaining baseline productivity	project reliability only slightly better than baseline while productivity falls below baseline

Figure 3. Multi-Project Analysis Study of Cleanroom in the SEL

The experiments to date convinced us that reading is a key, if not *the* key technical activity for

verifying and validating software work products. However, there has been little research focus on the development of reading techniques, with the possible exception of reading by stepwise abstraction, as developed by Harlan Mills.

The ultimate goal here is to understand the best way to read for a particular set of conditions. That is, we are not only interested in how to develop techniques for reading such documents as requirements documents, but under what conditions are each of the techniques most effective and how might they be combined in a method such as inspections to provide a more effective reading technology for the particular problem and environment.

The idea is to provide a flexible framework for defining the reading technology so that the definer of the technology for a particular project has the appropriate information for selecting the right techniques and method characteristics. Thus, the process definition will change depending on the project characteristics. For example, if the problem and solution are well understood, we might choose a waterfall process model; if a high number of omission faults are expected, we might emphasize a traceability reading approach embedded in design inspections; when embedding a traceability reading in design inspections, we might make sure a traceability matrix exists.

As stated in the introduction, we believe there are many factors that affect the way a person reads, e.g., the reviewer's role, the reading goals, the work product. Based upon these studies, we also believe that (1) techniques can be developed that will allow us better define how we should read, and (2) using these techniques, effectively embedded in the appropriate methods, can improve the effects of reading. For example, end-users read software requirements differently than do software testers, developers read for interface defects differently than they read for missing initialization. The more I know about what kinds of defects each of the views is most effective in tracking, the better I am able to promote and manipulate that kind of reading technique in the method I am using.

We need to improve the reading of all kinds of documents and more deeply understand the relationship between techniques and methods and the dimensions of both. For example, consider the following dimensions of a reading technique:

- Input object: Requirements, specification, design, code, test plan,...
- Output object: set of anomalies
- Approach: Sequential, path analysis, stepwise abstraction, ...
- Formality: Reading, correctness demonstrations, ...
- Emphasis: Fault detection, traceability, performance, ...
- Method: Walk-throughs, inspections, reviews, ...
- Consumers: User, designer, tester, maintainer, ...
- Product qualities: Correctness, reliability, efficiency, portability,...
- Process qualities: Adherence to method, integration into process,...
- Quality view: Assurance, control, ...

We have spent some energy trying to develop and evaluate reading techniques based upon the dimension and historical data. The goal is to define a set of reading technologies that can be tailored to the document being read and the goals of the organization for that document. The technology should be usable in existing methods, such as inspections.

2.4 Scenario-Based Reading

We have defined an approach to generating a family of reading techniques. It consists of building operational scenarios based upon combining two dimensions of the technique. An operational scenario requires the reader to (1) create an abstraction of the product (based on

one dimension) (2) answer questions based on the abstraction (based on another dimension). The choice of abstraction and the types of questions asked may depend on the document being read, the problem history of the organization or the goals of the organization. The scenarios try to take advantage of the dimensions of a technique (Figure 4).

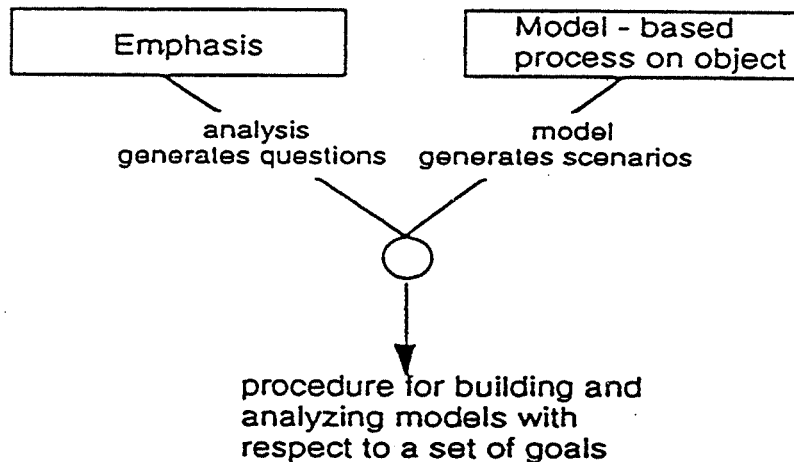


Figure 4. Building Focused Tailored Reading Techniques

Two different reading techniques, within the family, have been defined for requirements documents: defect based reading and perspective based reading

For our study, defect based reading was defined for reading SCR style documents. Defect based reading focuses on modeling different defect classes, creating three different scenarios based upon the data type consistency, safety properties, and ambiguity/missing information. The analysis questions were generated by combining/abstracting a set of checklist questions for requirements documents.

For our study, perspective based reading was defined for reading natural language requirements documents. Perspective-based reading focuses on different product customer perspectives, e.g., reading from the perspective of the software designer, the tester, or the end-user. The analysis questions were generated by focussing predominantly on various requirements type errors, e.g., incorrect fact, omission, ambiguity, and inconsistency.

To provide a little more detail into scenarios in general, and perspective based reading reading in particular, consider as an example, test-based reading.

Reading Procedure: For each requirement, make up a test or set of tests that will allow you to ensure that the implementation satisfies the requirement. Use your standard test approach and test criteria to make up the test suite. While making up your test suite for each requirement, ask yourself the following questions:

1. Do you have all the information necessary to identify the item being tested and to identify your test criteria? Can you make up reasonable test cases for each item based upon the criteria?
2. Is there another requirement for which you would generate a similar test case but would get a contradictory result?
3. Can you be sure the test you generated will yield the correct value in the correct units?
4. Are there other interpretations of this requirement that the implementor might make based upon the way the requirement is defined? Will this effect the test you made up?
5. Does the requirement make sense from what you know about the application and from

what is specified in the general description?

Each of the techniques aims at being (1) associated with the particular document (e.g., requirements) and notation (e.g., English text) in which the document is written, (2) tailorable, based upon the project and environment characteristics (3) detailed, in that it provides the reader a well-defined set of steps to follow, (4) specific, in that the reader has a particular purpose or goal for reading the document and the procedures support that goal, (5) focused, in that a particular technique provides a particular coverage of the document, and a combination of techniques provides coverage of the entire document, (6) studied empirically to determine if and when it is most effective.

Each of the techniques has been studied experimentally. The first series of experiments are aimed at discovering if scenario based reading is more effective than current practices. A second series will be used to discover under which circumstances each of the various scenario based reading techniques is most effective.

In the defect-based reading study, the goal was to analyze defect-based reading, ad-hoc reading and check-list based reading to evaluate and compare them with respect to their effect on fault detection effectiveness in the context of an inspection team from the viewpoint of quality assurance. The study was applied using graduate students at the University of Maryland. The requirements documents were written in the SCR notation. They were a water Level Monitoring System and a Cruise Control System. The experimental design is a blocked subject-project: Partial factorial design, replicated twice with a total of 48 subjects [Porter, Votta, Basili - 1995].

Major results were that (1) the defect-based readers performed better than ad hoc and checklist readers with an improvement of about 35%, (2) the defect-based reading procedures helped reviewers focus on specific fault classes but were no less effective at detecting other faults, and (3) checklist reading was no more effective than ad hoc reading.

Perspective-based reading is currently under study. The goal for the perspective-based reading evaluation was to analyze perspective-based reading, NASA's current reading technique to evaluate and compare them with respect to their effect on fault detection effectiveness in the context of an inspection team from the viewpoint of quality assurance. Two studies have been performed in the SEL environment using generic requirements documents written in English (ATM machine, Parking Garage) and NASA type functional specifications (Two ground support AGSS sub-systems). The experimental design is again a blocked subject-project using a partial factorial design. It has been applied twice, with a total of 25 subjects [Basili, Green, Laitenberger, Schull, Sorumgaard - 1995].

Preliminary indications for perspective based reading are also positive. Where there is any statistical significance, perspective-based reading appears to be more effective in uncovering defects and teams consisting of perspective based readers appear to do better than the standard reading techniques used.

3. CONCLUSION

Defect-based reading has been evaluated in experiments and has so far been shown to be superior to existing current practices. Perspective-based reading is being evaluated in experiments and the results so far appear promising.

Specifically we have run the experimental gamut from blocked subject-project experiments (reading vs. testing) to replicated projects (University of Maryland Cleanroom study) to a cased study (the first SEL Cleanroom study) to multi-project variation (the set of SEL Cleanroom projects) and now back to blocked subject project experiments (for scenario based reading). See Figure 5.

Scopes of Evaluation

		#Projects	
		One	More than one
# of Teams	One	3. Cleanroom (SEL Project 1)	4. Cleanroom (SEL Projects, 2,3,4, ...)
	More than one	2. Cleanroom at Maryland	1. Reading vs. Testing 5. Scenario Reading vs. ...

Figure 5. Series of Studies

In the future, we plan to replicate these experiments in many different environments. Various groups at different sites are already replicating some of the earlier experiments. Most of these are members of ISERN, the International Software Engineering Research Network, whose goal is specifically to perform and share the results of empirical studies.

We will continue to develop operational scenario reading techniques (e.g., design reading, etc.) and test their effectiveness in experiments. Future work also includes the consideration of tool support for the technologies developed.

4. REFERENCES

- Basili, V.R. (1985) Quantitative Evaluation of Software Methodology, Keynote Address, *First Pan Pacific Computer Conference*, Melbourne, Australia.
- Basili, V.R. (1989) Software Development: A Paradigm for the Future, *COMPSAC '89*, Orlando, Florida, pp.471-485.
- Basili, V.R., Caldiera, G., McGarry, F., Pajersky, R., Page, G., Waligora, S. (1992) The Software Engineering Laboratory--An Operational Software Experience Factory, *14th International Conference on Software Engineering*, Melbourne, Australia.
- Basili, V.R. and Green, S. (1994) Software Process Evolution at the SEL, *IEEE Software*, pp 58-66.
- Basili, V.R., Green, S., Laitenberger, O.U., Schull, F. and Sorumgaard, S. (1995) To be published) The Empirical Investigation of Perspective-Based Reading (in progress).
- Basili, V.R. and Rombach, H.D. (1988) The TAME Project: Towards Improvement-Oriented Software Environments, *IEEE Transactions on Software Engineering*, vol.14, no.6.
- Basili, V.R. and Selby, R. (1987) Comparing the Effectiveness of Software Testing Strategies, *IEEE Transactions on Software Engineering*, pp.1278-1296.

- Basili, V.R. and Weiss, D.M. (1984) A Methodology for Collecting Valid Software Engineering Data, *IEEE Transactions on Software Engineering*, pp.728-738.
- Currit, P.A., Dyer, M. and Mills, H.D. (1986) Certifying the Reliability of Software, *IEEE Transactions on Software Engineering*, vol.SE-12, pp. 3-11.
- Hetzl, W.C. (1972) An Experimental Analysis of Program Verification Problem Solving Capabilities as They Relate to Programmer Efficiency, *Computer Personnel*, vol.3, pp.10-15.
- Linger, R.C., Mills, H.D. and Witt, B.I. (1979) *Structured Programming: Theory and practice*, Reading, MA: Addison-Wesley.
- Myers, G.J. (1978) A Controlled Experiment in Program Testing and Code Walkthroughs Inspections, *Communications ACM*, pp.760-768.
- Porter, A.A., Votta, L.G. and Basili, V.R. (1995) Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment, *IEEE Transactions on Software Engineering*, vol.21, no.6, pp.563-575.
- Selby, R., Basili, V.R. and Baker, T. (1987) Cleanroom Software Development: An Empirical Evaluation, *IEEE Transactions on Software Engineering*, pp 1027-1037.

58-61
415774

The Empirical Investigation of Perspective-Based Reading

Victor R. Basili¹, Scott Green², Oliver Laitenberger³,
Forrest Shull¹, Sivert Sørungård⁴, Marvin V. Zelkowitz¹

¹ Computer Science Department/
Institute for Advanced Computer Studies
University of Maryland, College Park, MD, 20742
{basili, fshull, mvz}@cs.umd.edu

² NASA Goddard Space Flight Center
Code 552.1
Greenbelt, MD, 20771
scott.green@gsfc.nasa.gov

³ AG Software Engineering
Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
67653 Kaiserslautern
Germany
laitenbe@informatik.uni-kl.de

⁴ The Norwegian Institute of Technology
The University of Trondheim
UNIT/NTH-IDT
O.S. Bragstads plass 2E
Trondheim, N-7034
Norway
sivert@idt.unit.no

Abstract

We consider reading techniques a fundamental means of achieving high quality software. Due to the lack of research in this area, we are experimenting with the application and comparison of various reading techniques. This paper deals with our experiences with Perspective-Based Reading (PBR), a particular reading technique for requirements documents. The goal of PBR is to provide operational scenarios where members of a review team read a document from a particular perspective (e.g., tester, developer, user). Our assumption is that the combination of different perspectives provides better coverage of the document than the same number of readers using their usual technique.

This work was supported in part by NASA grant NSG-5123 and UMIACS.

To test the efficacy of PBR, we conducted two runs of a controlled experiment in the environment of the National Aeronautics and Space Administration / Goddard Space Flight Center (NASA/GSFC) Software Engineering Laboratory (SEL), using developers from the environment. The subjects read two types of documents, one generic in nature and the other from the NASA domain, using two reading techniques, PBR and their usual technique. The results from these experiments, as well as the experimental design, are presented and analyzed. When there is a statistically significant distinction, PBR performs better than the subjects' usual technique. However, PBR appears to be more effective on the generic documents than on the NASA documents.

1. Introduction

The primary goal of software development is to generate systems that satisfy the user's needs. However, the various documents associated with software development (e.g., requirements documents, code and test plans) often require continual review and modification throughout the development lifecycle. In order to analyze these documents, reading is a key, if not *the* key technical activity for verifying and validating software work products. Methods such as inspections (Fagan, 1976) are considered most effective in removing defects during development. Inspections rely on effective reading techniques for success.

Reading can be performed on all documents associated with the software process, and can be applied as soon as the documents are written. However, except for reading by step-wise abstraction (Linger, 1979) as developed by Harlan Mills, there has been very little research focused on the development of reading techniques. Most efforts have been associated with the methods (e.g., inspections, walk-throughs, reviews) surrounding the reading technique. In general, techniques for reading particular documents, such as requirements documents or test plans, do not exist. In cases where techniques do exist, the required skills are neither taught nor practiced. In the area of programming languages, for example, almost all effort is spent learning how to *write* code rather than how to *read* code. Thus, when it comes to reading, little exists in the way of research or practice.

In the Software Engineering Laboratory (SEL) environment, we have learned much about the efficacy of reading and reading-based approaches through the application and evaluation of methodologies such as Cleanroom. We are now part of a group (ISERN¹) that has

undertaken a research program to define and evaluate software reading techniques to support the various review methods for software development.

In this paper, we use the following convention to differentiate a "technique" from a "method": A technique is a series of steps, producing some desired effect, and requiring skilled application. We define a method as a management procedure for applying techniques.

1.1 Experimental Context: Scenario-Based Reading

In our attempt to define reading techniques, we established several goals:

- The technique should be associated with the particular document (e.g., requirements) and the notation in which the document is written (e.g., English text). That is, it should fit the appropriate development phase and notation.
- The technique should be tailorable, based upon the project and environment characteristics. If the problem domain changes, so should the reading technique.
- The technique should be detailed, in that it provides the reader with a well-defined process. We are interested in usable techniques that can be repeated by others.
- The technique should be specific in that each reader has a particular purpose or goal for reading the document and the procedures support that goal. This can vary from project to project.
- The technique should be focused in that a particular technique provides a particular coverage of the document, and a combination of techniques provides coverage of the entire document.
- The technique should be studied empirically to determine if and when it is most effective.

To this end, we have defined a set of techniques, which we call proactive process-driven scenarios, in the form of algorithms that readers can apply to traverse the document with a particular emphasis. Because the scenarios are focused, detailed, and specific to a particular emphasis or viewpoint, several scenarios must be combined to provide coverage of the document.

We have defined an approach to generating a family of reading techniques based upon operational scenarios, illustrated in Figure 1. An operational scenario requires the reader to first create an abstraction of the product, and then answer questions based on the abstraction. The choice of abstraction and the types of questions asked may depend on the document being read, the problem history of the organization or the goals of the organization.

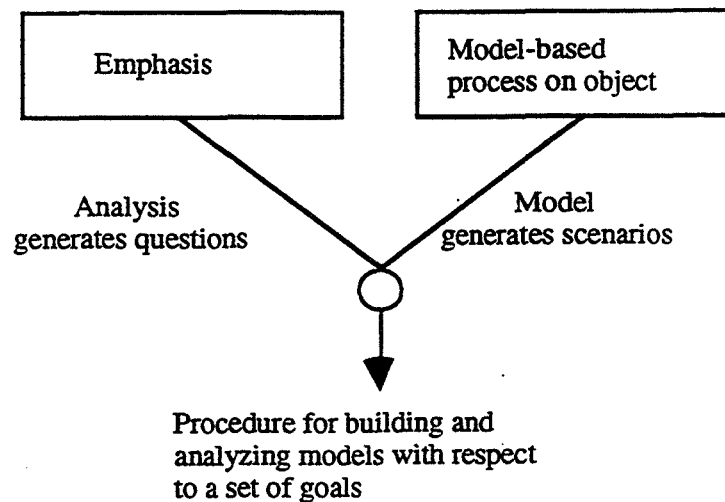


Figure 1. Building focused, tailored reading techniques.

So far, two different scenario-based reading techniques have been defined for requirements documents: perspective-based reading and defect-based reading.

Defect-based reading was the subject of an earlier set of experiments in this series. Defect-based reading was defined for reading SCR (Software Cost Reduction) style documents (Heninger, 1980), and focuses on different defect classes, e.g., missing functionality and data type inconsistencies. These create three different scenarios: data type consistency, safety properties, and ambiguity/missing information. An experimental study (Porter, 1995) was undertaken to analyze defect-based reading, ad hoc reading and checklist-based reading to evaluate and compare them with respect to their effect on defect detection rates. Major results were that (1) scenario readers performed better than ad hoc and checklist readers with an improvement of about 35%, (2) scenarios helped reviewers focus on

specific defect classes but were no less effective at detecting other defects, and that (3) checklist reading was no more effective than ad hoc reading.

However, the experiment discussed in this paper is concerned with an experimental validation of perspective-based reading, and so we treat it in more detail in the next section.

1.2 Perspective-Based Reading

Perspective-based reading (PBR) focuses on the point of view or needs of the customers or consumers of a document. In this type of scenario-based reading, one reader may read from the point of view of the tester, another from the point of view of the developer, and yet another from the point of view of the user of the system. To provide a proactive scenario, each of these readers produces some physical model which can be analyzed to answer questions based upon the perspective. The team member reading from the perspective of the tester would design a set of tests for a potential test plan and answer questions arising from the activities being performed. Similarly, the team member reading from the perspective of the developer would generate a high level design, and the team member representing the user would create a user's manual. Each scenario is focused on one perspective. The assumption is that the union of the perspectives provides sufficient coverage of the document but does not cause any particular reader to be responsible for everything.

This work on PBR was conducted within the confines of the NASA/GSFC Software Engineering Laboratory. The SEL, started in 1976, has been developing technology aimed at improving the process of developing flight dynamics software within NASA/GSFC. This class of software is typically written in any of several programming languages, including FORTRAN, C, C++, and Ada. Systems can range from 20K to 1M lines of source code, with development teams of up to 15 persons working over a one to two year period.

Assume we embed these requirements reading scenarios in a particular method. It then becomes the role of the method to determine which scenarios to apply to the document, how many readers will play each role, etc. This could be done by assuming, as entry criteria, that the method has available to it the anticipated defect class distribution, together with knowledge of the organization's ability to apply certain techniques effectively. Note that embedding focused reading techniques in a method such as inspections provides more

meaning to the "team" concept. That is, it gives the readers different views of the document, allowing each of the readers to be responsible for their own view, with the union of the readers providing greater coverage than any of the individual readers.

Consider, as an example, the procedure for a reader applying the test-based perspective:

Reading Procedure: For each requirement, make up a test or set of tests that will allow you to ensure that the implementation satisfies the requirement. Use your standard test approach and test criteria to make up the test suite. While making up your test suite for each requirement, ask yourself the following questions:

1. Do you have all the information necessary to identify the item being tested and to identify your test criteria? Can you make up reasonable test cases for each item based upon the criteria?
2. Is there another requirement for which you would generate a similar test case but would get a contradictory result?
3. Can you be sure the test you generated will yield the correct value in the correct units?
4. Are there other interpretations of this requirement that the implementor might make based upon the way the requirement is defined? Will this effect the test you made up?
5. Does the requirement make sense from what you know about the application and from what is specified in the general description?

These five questions form the basis for the approach the test-based reader will use to review the document.

We developed two different series of experiments for evaluating scenario-based techniques. The first series of experiments are aimed at discovering if scenario-based reading is more effective than current practices. This paper's goal is to analyze perspective-based reading and the current NASA SEL reading technique to evaluate and compare them with respect to their effect on fault detection effectiveness. It is expected that other studies will be run in

different environments using the same artifacts where appropriate. A second series, to be undertaken later, will be used to discover under which circumstances each of the various scenario-based reading techniques is most effective.

1.3 Experimental Plan

Our method for evaluating PBR was to see if the approach was more effective than the approach people were already using for reading and reviewing requirements specifications. Thus, it assumes some experience in reading requirements documents on the part of the subjects. More specifically, the current NASA SEL reading technique (SEL, 1992) had evolved over time and was based upon recognizing certain types of concerns which were identified and accumulated as a set of issues requiring clarification by the document authors, typically the analysts and users of the system.

To test our hypotheses concerning PBR, a series of partial factorial experiments were designed, where subjects would be given one document and told to discover defects using their current method. They would then be trained in PBR and given another document in order to see if their performance improved. We were initially interested in several outcomes:

1. Would individual performances improve if each individual used one of the PBR (designer, tester, user) scenarios in order to find defects?
2. If groups of individuals (such as during an inspection meeting) were given unique PBR roles, would the collection of defects be different than if each read the document in a similar way?
3. Are there characteristic differences in the class of defects each scenario uncovered?

While we were interested in the effectiveness of PBR within our SEL environment, we were also interested in the general applicability of the technique in environments different from the flight dynamics software that the SEL generally builds. Thus two classes of documents were developed: a domain-specific set that would have limited usefulness outside of NASA, and a generic set that could be reused in other domains.

For the NASA flight dynamics application domain, two small specifications derived from an existing set of requirements documentation were used. These specification documents, seeded with classes of errors common to the environment, were labeled NASA_A and NASA_B. For the generic application domain, two requirements documents were developed and seeded with known classes of errors. These applications included an automated parking garage control system, labeled PG, and an automated bank teller machine, labeled ATM.

1.4. Structure of this Paper

In section 2, we discuss how we developed a design for the experiment outlined above. Major issues concerning constraints and threats to validity are described in order to highlight some of the tradeoffs made. We also include a short overview of how the experiment was actually carried out.

Section 3 presents the statistical analysis of the data we obtained in the experiment. The section examines individual results and team results. In each of these parts, we look at the results from both experiment runs, within documents and within domains.

Section 4 is an interpretation of the results of the experiment, but without the rigor of a formal statistical approach. The presentation is again divided into individual results and team results, with concentration on what effect the differences between the two runs of the experiment had in terms of results.

Section 5 summarizes our experiences regarding designing and carrying out the experiment.

2. Design of the Experiment

In this section, we discuss various ways of organizing the individual subjects and the instrumentation of the experiment to test various hypotheses. Two runs of the experiment were conducted. Due to the experiences gained in the initial run, some modifications were introduced in its replication. Differences between the two runs of the experiment will be pointed out where appropriate.

For both experiments, the population was software developers from the NASA SEL environment. The selection of subjects from this sample was not random, since everyone in the population could not be expected to be willing or have opportunity to participate. Thus, all subjects were volunteers, and we accepted everyone who volunteered. Nobody participated in both runs of the experiment.

2.1 Hypotheses

We formulated our main question in the form of the following two hypotheses, where H_0 is the null-hypothesis and H_a is the alternative hypothesis:

H_0 *There is no significant difference in the defect detection rates of teams applying PBR as compared to teams using the usual NASA technique.*

H_a *The defect detection rates of teams applying PBR are significantly higher as compared to teams using the usual NASA technique.*

Our hypotheses are focused on the performance of teams, but we will also analyze the results for the individual performance of the subjects. We make no assumptions at this level regarding the validity of the hypotheses when changing important factors such as subjects, and documents. The constraints relevant for this particular experiment will be explicitly discussed throughout this section, as will the generalizability of the results of the experiment.

2.2 Factors in the Design

In designing the experiment, we had to consider what factors were likely to have an impact on the results. Each of these factors will cause a rival hypothesis to exist in addition to the hypotheses we mentioned previously. The design of the experiment has to take these factors, called *independent variables*, into account and allow each of them to be separable from the others in order to allow for testing a causal relationship to the defect detection rate, the *dependent variable* under study.

Below we list the independent variables, which we identify according to how they can be manipulated. Some of them can be controlled during the course of the experiment, while some are strictly functions of time, and still others are not even measurable.

- **Controllable variables:**

- **Reading technique:** We have two alternatives: One is the technique we have developed, PBR, and the other is the technique currently used for requirements document review in the NASA SEL environment, which we refer to as the "usual" technique.
- **Requirements documents:** For each task to be carried out by the subjects, a requirements specification is handed out to be read and reviewed. The document will presumably have an impact on the results due to differences in size, domain and complexity.
- **Perspective:** For PBR, a subject can take one of three perspectives as previously described: Designer, Tester or User.

- **Measurable variables:**

- **Replication:** This nominal variable is not one we can manipulate, but we need to be aware of its presence because there may be differences in the data from the two experiment runs that may be the result of changes to documents, training sessions or experimental conditions.
- **Round within the replication:** For each experiment, every subject is involved in a series of treatments and tasks or observations. The results from similar tasks may differ depending on when they take place.

- **Other factors identified:**

- **Experience:** The experience of each subject is likely to have an impact on the defect detection rate.
- **Task sequence:** Reading the documents in a sequence may have an influence on the results. This may be a learning effect due to the repetitive reading of multiple documents.
- **Environment:** The particular environment in which the experiment takes place may have an impact on how well the subjects perform. In this experiment, this effect cannot be separable from effects due to replication.

There will also be other factors present that may have an impact on the outcome of the experiment, but that are hard to measure and control. These will be discussed in Section 2.5. This section will also cover the last two factors mentioned above: Task Sequence (in the literature referred to as "effects due to testing") and Environment.

2.3 Constraints and Limitations

In designing the experiment we took into account various constraints that restrict the way we could manipulate the independent variables. There are basically two factors that constrain the design of this experiment:

- **Time:** Since the subjects in this experiment are borrowed from a development organization, we could not expect to have them available for an indefinite amount of time. This required us to make the experiment as time-efficient as possible without compromising the integrity of the design.
- **Subjects:** For the same reasons as stated above, we could not expect to get as many subjects as we would have liked. This required us to be cautious in the design and instrumentation in order to generate as many useful data points as possible.

Specifically, we knew that we could expect to get between 12 and 18 subjects for two days on any run of the experiment.

Another factor that we had to deal with is that we had to provide some potential benefit to the subjects since their organization was supporting their participation. Training in a new approach provided some benefit for their time. This had an impact on our experimental design because we had to treat people equally as far as the training they received.

2.4 Choosing a Design

Due to the constraints, we found that constructing real teams of (three) reviewers to work together in the experiment would take too much time for the resulting amount of data points. This decision was supported by similar experiments (Parnas, 1985) (Porter, 1995) (Votta, 1993), where the team meetings were reported to have little effect; the meeting gain was outweighed by the meeting loss. However, the team is an important unit in the review process, and PBR is team-oriented in that each reviewer has a responsibility that is not

shared by other reviewers on the team. Thus our reviewers did not work together in teams during the course of the experiment. Instead we conducted the experiment based on individual tests, and then used these individual results to construct hypothetical teams after the experiment was completed.

The tasks performed by the subjects consisted of reading and reviewing a requirements specification document, and recording the identified defects on a form. The treatments, which had the purpose of manipulating one or more of the independent variables, were aimed at teaching the subjects how to use PBR. There were four possible ways of arranging the order of tasks and treatments for a group of subjects:

1. Do all tasks using the usual technique.
2. Do pre-task(s) with the usual technique, then teach PBR, followed by post-task(s) using PBR.
3. Start by teaching PBR, then do some tasks with the PBR technique, followed by tasks using the usual technique.
4. Start by teaching PBR, then do all tasks using PBR.

Option 3, where the subjects first use PBR and then switch to their usual technique, was not considered an alternative because their recent knowledge in PBR may have undesirable influences on the way they apply their usual technique. The opposite may also be true, that their usual technique has an influence on the way they apply PBR, but that is a situation we cannot control because the subjects already know their usual technique. Thus, this becomes more a problem in terms of external validity.

All documents reviewed by a subject must be different. If a document was reviewed more than once by the same subject, the results would be disturbed by the subject's non-erasable knowledge about defects found in previous readings. This meant that we had to separate the subjects into two groups - one reading the first document and one reading the second in order to be able to compare a PBR and a usual reading of a document.

Based on the constraints of the experiment, each subject would have time to read and review no more than four documents: two from the generic domain, and two from the NASA domain. In addition, we needed one sample document from each domain for training purposes. We ended up providing the following documents:

- **Generic:**
 - Automatic teller machine (ATM) - 17 pages, 29 seeded defects.
 - Parking garage control system (PG) - 16 pages, 27 seeded defects.
- **NASA:**
 - Flight dynamics support (A) - 27 pages, 15 seeded defects
 - Flight dynamics support (B) - 27 pages, 15 seeded defects
- **Training:**
 - Video rental system - 14 pages, 16 seeded defects
 - NASA sample - 9 pages, 6 seeded defects

Since we have sets of different documents and techniques to compare, it became clear that a variant of factorial design would be convenient for this experiment. Such a design would allow us to test the effects of applying both of the techniques on both of the relevant documents. We found that a full factorial design would be inappropriate for two reasons. First, a full factorial design would require some subjects to apply the ordering of techniques that we previously argued against. Secondly, such a design seemed hard to conduct because it would require each subject to use all three perspectives at some point. This would require an excessive amount of training, and perhaps even more important, the perspectives would likely interfere with each other, causing an undesirable learning effect.

The use of control groups to assess differences in documents and learning effect appeared to bear an unreasonable cost, since the use of such groups would decrease the remaining number of data points available for analyzing the difference between the techniques. The low number of data points might result in data that would be heavily biased due to individual differences in performance. Based on the cost and the fact that previous related experiments (Porter, 1995) showed that effects of learning were not significant, we chose not to use control groups. This decision also made the experiment more attractive in terms of getting subjects, since they would all receive the same amount and kind of training.

	Group 1			Group 2			
	D	T	U	D	T	U	
NASA technique	Training			Training			First day
	NASA A			NASA B			
	Training			Training			
	ATM			PG			
PBR technique	Teaching of PBR						Second day
	Training			Training			
	PG			ATM			
	Training			Training			
	NASA B			NASA A			

Figure 2. Design of the experiment.

We blocked the design on technique, perspective, document and reading sequence in order to get an equal distribution of the values of the different independent variables. Thus we ended up with two groups of subjects, where each group contains three subgroups, one for each perspective (see Figure 2). The number of subjects was about the same for the two experiments (12-14 subjects).

2.5 Threats to Validity

Threats to validity are factors beyond our control that can affect the dependent variables. Such threats can be considered unknown independent variables causing uncontrolled rival hypotheses to exist in addition to our research hypotheses. One crucial step in the experimental design is to minimize the impact of these threats.

We have two different classes of threats to validity: threats to *internal* validity and threats to *external* validity. Threats to internal validity constitute potential problems in the interpretation of the data from the experiment. If the experiment does not have a minimum internal validity, we can make no valid inference regarding the correlation between variables. On the other hand, the level of external validity tells us nothing about whether the data is interpretable, but is an indicator of the generalizability of the results. Depending on the external validity of the experiment, the data can be assumed to be valid in other populations and settings.

The following five threats to internal validity (Campbell, 1963) are discussed in order to reveal their potential interference with our experimental design:

- **History:** We need to consider what the subjects did between the pretests and posttests. In addition to receiving a treatment where they were taught a new reading technique, there may have been other events outside of our control that had an impact on the results. The subjects were instructed not to discuss the experiment or otherwise do anything between the tests that could cause an unwanted effect on the results.
- **Maturation:** This is the effect of processes taking place within the subjects as a function of time, such as becoming tired or bored. But it may also be intellectual maturation, regardless of the experimental events. For our experiment, the likely effect would be that tests towards the end of the day tend to get worse results than they would normally. We provided generous breaks between sessions to suppress this effect.
- **Testing:** Getting familiar with the tests may have effects on subsequent results. This threat has several components, including becoming familiar with the specifications, the technique, or the testing procedures. We tried to overcome unwanted effects by providing training sessions before each test where the subjects could familiarize themselves with the particular kind of document and technique. Also, the subjects received no feedback regarding their actual defect detection success during the experiment, as this would presumably increase the learning effect. Related experiments have reported that effects due to repeated testing are not significant (Porter, 1995).
- **Instrumentation:** These effects are basically due to differences in the way of measuring scores. Our scores were measured by two people independently, and then discussed in order to resolve any disagreement consistently. Thus this effect is not relevant to us.
- **Selection:** Subjects may be assigned to their treatment groups in various ways. In our case there was a difference between the two experiment runs. In the first one, the subjects were assigned roles for PBR based on their normal work in the NASA environment in order to match roles as closely as possible. This was only minimally successful since the sample was not an even mix of people representing the various roles. However, for the replication, the subjects were randomized. Thus effects due to selection may be somewhat relevant for the first experiment, but not for the replication. Since PBR assumes the reviewers in a team use their usual perspectives, the random assignment used in the experiment would presumably lead to an underestimation of the improvement caused by PBR.

Another threat to validity is the possibility that the subjects ignore PBR when they are supposed to use it. In particular, there is a danger that the subjects continue to use their usual technique. This need not be the result of a deliberate choice from the subject, but may simply reflect the fact that people unconsciously prefer to apply existing skills with which they are familiar. The only way of coping with this threat is to provide enhanced training sessions and some sort of control or measure of conformance to the assigned technique.

Threats to external validity imply limitations to generalizing the results. The experiment was conducted with professional developers and with documents from an industrial context, so these factors should pose little threat to external validity. However, the limited number of data points is a potential problem which may only be overcome by further replications of the experiment. Other threats to external validity pertinent to the experimental design include (Campbell, 1963):

- **Interaction of testing and treatment:** A pretest may affect the subject's sensitivity of the experimental variable. Both of our groups receive similar pretests and treatments, so this effect may be of concern to us.
- **Interaction of selection and treatment:** Selection biases may have different effects due to interaction with the treatment. One factor we need to be aware of is that all our subjects were volunteers. This may imply that they are more prone to improvement-oriented efforts than the average developer - or it may indicate that they consider the experiment an opportunity to get away from normal work activities for a couple of days. Thus, the effects can strike in either direction. Also, all subjects had received training in their usual technique, a property that developers from other organizations may not possess.
- **Reactive effects:** These effects are due to the experimental environment. Here we have a difference between the two runs of the experiment. In the initial run, the testing was done in the subjects' usual work environment. The subjects received their training in groups, and then returned to their own workspace for the test. For the replication, the experiment was conducted in an artificial setting away from the work environment, similar to a classroom exercise. This may influence the external validity of the experiment, since a non-experimental environment may cause different results.

There are also a number of other possible but minor threats. One of these is the fact that the subjects knew they were part of an experiment. They knew that the purpose of the experiment was to compare reading techniques, and they probably were able to surmise our

expectations with respect to the results even if not stated explicitly. However, these aspects are difficult to eliminate in experiments where subjects are trained in one technique while the comparison technique is assumed to be known in advance. A design where they receive equal training in two techniques would be more likely to hide these effects.

2.6 Preparation and Conduction

We wanted the two experiment runs to be as similar as possible in order to avoid difficulties in combining the resulting data, but some changes between the runs were still necessary. We began preparing for the second run by reviewing all documents and forms in order to improve them from an experimental viewpoint. We had some comments from the first experiment run that were helpful in this process. The changes were minor, and most were directed towards language improvement. We changed the seeded defects in three places in one of the generic documents due to a refined and deeper insight into what we would consider a defect. There were some changes to the forms, scenarios and defect classification as well, but again the changes were made to make the documents easier to use and understand.

For the NASA documents, the changes were more fundamental. For the first experiment run, comments from the participants indicated that the documents were too large and complex. We decided to make them shorter and simpler for the second experiment run. As a side effect of this change, the total number of defects in the NASA documents was reduced. However, the types and distribution of seeded defects remained similar.

The basic schedule for conducting the experiment remained unchanged. Each experiment run lasted for two whole work days, with one day off in-between. The number and order of document reviews were also the same for both experiments, but the time allowed for each review was modified. For the first experiment run, the maximum time for one document was three hours. However, for the generic documents, only one person used more than two hours (140 minutes), so under the more controlled environment of the second experiment run, we felt safe lowering the maximum time to two hours.

Another important change resulted from the comments we received from the first experiment run, regarding the training sessions. The initial run included training sessions only for the generic documents, but the subjects felt training for the NASA documents was warranted as well. Therefore in the second experiment run, we had training sessions

before each document review. For this purpose we generated two sample documents that were representative of the NASA and generic domains.

After the second run of the experiment, we marked all reviews with respect to their defect detection rate. This was measured as the percentage of the seeded defects that was found by each reviewer. We did not consider any other measures such as false positives. Based on the defects found by the reviewers, we also refined our understanding of the defects present in the set of documents. After several iterations of discussion and re-marking, we arrived at a set of defect lists that were considered representative of the documents. Since these lists were slightly different from the lists that were used in the first experiment, we re-marked all the reviews from the first experiment in order to make all results consistent.

3. Statistical Analysis

We ran the experiment twice, in November 1994 (hereafter referred to as the "1994 experiment") and in June 1995 (hereafter referred to as the "1995 experiment"). In the 1994 experiment, we had twelve subjects read each document, six using the usual technique and six using PBR. The six using PBR were distributed equally among the three perspectives. In the 1995 experiment, we had thirteen subjects who read each document, although a fourteenth volunteer unfamiliar with the NASA domain also read the generic documents only.

After the two experiment runs, we have a substantial base of observations from which to draw conclusions about PBR. This task is complicated, however, by the various sources of extraneous variability in the data. Specifically, we identify four other variables (besides the reading technique) which may have an impact on the detection rate of a reviewer: the experiment run within which the reviewer participated, the problem domain, the document itself, and the reviewer's experience.

We attempted to measure reviewer experience via questionnaires used during the course of the experiment: a subjective question asked each reviewer to rate on an ordinal scale his or her level of comfort using such documents, and objective questions asked how many years the reviewer had spent in each of several roles (analyst, tester, designer, manager). However, for any realistic measurement scale, most reviewers tended to clump together toward the middle of the range, with relatively few outliers in either direction. Thus we seem to have a relatively homogeneous sample with respect to this variable. While good

from an experimental viewpoint, this unfortunately means that our data set does not allow for a meaningful test of the effect of reviewer experience, and we are forced to defer an investigation of the interaction between reading technique and experience until such time as we can get more data points. For this reason, reviewer experience will not appear as a potential effect in any of our analysis models.

Technique, experiment run, and document are represented by nominal-scale variables used in our models, where appropriate. The domain is taken into account by performing a separate analysis for each of the generic and NASA problem domains. However, we are also careful to note that there are variables that our statistical analysis cannot measure. Perhaps most importantly, an influence due to a learning effect would be hidden within the effect of the reading technique. The full list of these threats to validity is found in Section 2.5, and any interpretation of results must take them into account.

Section 3.1 presents the details of the effect on individual scores. Section 3.2 presents the analysis strategy for team data. Finally, Section 3.3 takes an initial look at the analysis with respect to the reviewer perspectives. In each section, we present the general analysis strategy and some details on the statistical tests, followed by the statistical results and some interpretation of their meaning. We address the significance of our results taken as a whole in Section 4.

3.1 Analysis for Individuals

Although it was not part of our main hypothesis, which focuses on team coverage, we wanted to see if the difference in focus between the usual technique and PBR would have some effect on individual detection rates. We therefore went through an analysis of individual scores.

We were also careful, however, to test for effects from sources of variation other than the reading technique. For this reason, our analysis proceeds in a "bottom-up" manner. That is, we begin with several small data sets that we know to be homogeneous. Each session of the experiment was run under controlled conditions to eliminate differences within the sessions that might have an effect on reviewers' detection rates; the scores of reviewers reading the same document within the same replication are therefore comparable. Thus we begin our analysis with homogeneous data sets (4 documents - 2 NASA and 2 generic - over 2 runs, so 8 in total) which we will use as the primary building blocks of our analysis.

Starting from these data sets, we looked for features in common between the data sets. We identified subsets of the data which were expected to be more homogeneous than the data as a whole; the aim was to exploit this homogeneity to achieve stronger statistical results. For example, we took into account the fact that all of the detection rates for each reviewer are highly correlated, but we also identified other such blocks (e.g., the data for each problem domain within the experiment). As we looked at larger data sets in order to draw more general conclusions, we also took pains to make sure that the data within each set were still comparable. Figure 3 illustrates the direction of our analysis, and includes the sizes of the data sets.

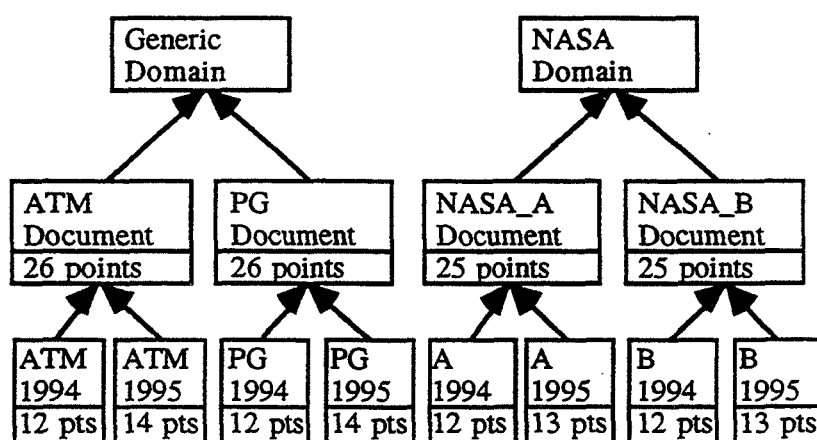


Figure 3. Breakdown of the statistical analysis, with number of data points.

3.1.1 Analysis Strategy Within Documents

Our initial analysis examined each document used in the experiment for significant differences in performance based on the use of reading technique. We used the ANOVA test since we were testing a model of the effects containing multiple potential sources of variation. To begin with, our model of the effects contained a nominal variable to signify the reading technique used (usual or PBR).

The data for each document is composed of the independent data sets from the two experiment runs, and so it was necessary to be alert to the possibility that changes from one run to the next could have an impact on the reviewers' detection rates. For both of the pairs of documents, we combined the data for the document and introduced a nominal variable

(with two levels: 1994 and 1995) into our model to describe the experiment run in which the reviewer read the document.

We measured the lack of fit error (an estimate of the error variance) for the model on each document. In no case was there a significant lack of fit error, so it did not seem likely that we could gain any better fit to the data by introducing variations on the variables, such as testing for interaction effects (SAS, 1989).

We also tested whether each of the variables independently was significant (i.e., whether the effect of each variable, apart from the other variables in the model, had a significant effect on reviewer detection rate).

The ANOVA test makes a number of assumptions, which we were careful to fulfill: The dependent variable is measured on a ratio scale, and the independent variables are nominal. Observations are independent. The values tested for each level of the independent variables are normally distributed (we confirmed this with the Shapiro-Wilk W Test). Also, the test assumes that variance between samples for each level of the independent variables is homogeneous. However, we note that the test is robust against violations of this last assumption for data sets such as ours in which the number of subjects in the largest treatment group is no more than 1.5 times greater than the number of subjects in the smallest (Hatcher, 1994). The test also assumes that the sample must be obtained through random sampling; this is a threat to the validity of our experiment, as we must rely on volunteers for our subjects (see Section 2.5, "Selection" and "Interaction of selection and treatment").

3.1.2 Results Within Documents

In our case the hypotheses of the ANOVA test take the following form:

H₀: The specified model (which contains variables to signify the experiment run and reading technique) has no significant power in predicting the value of the dependent variable (detection rate).

H_a: The model as a whole is a significant predictor of detection rate.

Level of significance: $\alpha = 0.05$

The ANOVA test also allows testing the effect of each individual variable.

The Least Squares Means (LSM) of the detection rates for reviewers using each of the techniques are given in Table 1, followed by the results of the tests for significance. The LSM values in effect allow an examination of the means for the groups using each of the reading techniques while holding the difference due to experiment run constant. This is followed by the p-values resulting from the statistical tests for significance; a p-value of less than 0.05 provides evidence that either the whole model or the individual variable is a significant predictor of detection rate and are indicated in boldface. The R^2 value for the model is also included as a measure of the amount of variation in the data that is accounted for by the model.

For all documents except NASA_B, the LSM detection rate for PBR reviewers is slightly higher than for reviewers using their usual technique. However, only for the ATM document was the difference statistically significant. For all other documents, reviewers using the two techniques did roughly the same, and any differences between their average scores can be attributed to random effects alone. Both NASA documents had a very significant effect due to experiment run, which was not surprising, given the large changes made to improve the documents between runs; however, there was also a significant and unexpected effect due to experiment run for the PG document as well. The significance of such differences due to experiment run is addressed in Section 4.

Document	PBR LSM	USUAL LSM	Whole Model p-value	Technique p-value	Replicat- ion p-value	R^2
ATM	30.8	21.4	0.0904	0.0316	0.6299	0.19
PG	26.8	24.5	0.0457	0.5977	0.0174	0.24
NASA_A	36.8	26.6	0.0001	0.1516	0.0001	0.56
NASA_B	28.3	34.5	0.0021	0.5044	0.0005	0.43

Table 1. Effects on individual scores for each document.

3.1.3 Analysis Strategy Within Domains

The second level of detail which we analyzed was the level of problem domains. That is, we examined what trends could be observed within the generic documents or within the NASA documents, while realizing that such trends may not necessarily apply across such different domains. For each domain, we tested whether each reviewer scored about the

same when reviewing documents with PBR as when using the usual technique, or if there was in fact a significant effect due to reading technique.

To accomplish this, we made use of the MANOVA (Multivariate ANOVA) test with repeated measures, an extension of the ANOVA which measures effects across multiple dependent variables (here, the scores on each of the two documents) with longitudinal data sets (i.e. data sets in which each subject is represented by multiple data points).

The domain data sets contain two scores for each subject, one for each document within the domain. Although repeated measures tests usually refer to multiple treatments over time, here we treat the scores on each document as the scores from repeated treatments, which we distinguish with the nominal variable "Document". We divide the reviewers into two groups, and use another nominal variable in order to distinguish to which group each reviewer belonged: Group I applied PBR to Document A and the usual technique to Document B, and Group II read the documents in the opposite fashion. If the interaction between these two variables is significant, we can conclude that the reading technique a reviewer applied to each document had a significant effect on the reviewer's detection rate. If the interaction is not significant, then reviewers tended to perform about the same on the two documents, regardless of the technique applied to each. Aside from reading technique and document, we again want to account for any significant effects due to the experiment run, and also test for interaction effects between this variable and the others.

The MANOVA test with Repeated Measures makes certain assumptions about the data set. As with the ANOVA test, we again fulfill requirements about the measurement scales of the dependent and independent variables, the independence of observations, and the underlying distribution of the sample. We have the same threat to validity resulting from the assumption of random samples as was discussed for the ANOVA test. However, it is also assumed that the dependent-variable covariance matrix for a given treatment group should be equal to the covariance matrix for each of the remaining groups. Fortunately, the type I error rate is relatively robust against typical violations of this assumption; however, the power of the test is somewhat attenuated (Hatcher, 1994).

3.1.4 Results Within Domains

Using the data from each of the documents within a domain, we use the MANOVA test to detect how reviewer rates change from one document to the next, and attribute these

changes to factors in our model. As we did with the ANOVA test, we test whether each of the variables in our model (the documents themselves, the reading technique used on each document, the experiment run, and all appropriate interactions) are significant predictors of the change in detection rates.

H₀: The specified variable has no significant effect in predicting scores across the two documents.

H_a: The variable is a significant predictor of scores across the documents.

Level of significance: $\alpha = 0.05$

The results are summarized in Table 2, where each column gives the p-value for each of the effects. A p-value of less than 0.05 provides an indication that the variable is a significant predictor of the change in reviewer detection rates across documents, and appears in bold. The effect due to the reading technique is measured indirectly by the "Group" variable: Group I read Document A with the PBR technique and Document B with the usual technique; Group II read the documents in the reverse fashion. As can be seen from the "Document" column, there was no significant difference between the mean detection rates for the two documents within a domain. Crossed terms represent tests for interaction effects; for example, the column labeled "Document * Replication" tests if the mean difference in reviewers' scores on each of the documents was significantly effected by the experiment run in which they took part. Thus, even though the NASA documents were changed drastically between runs, because the two documents were roughly comparable in difficulty within both experiment runs, there is no significant effect here for the NASA domain. Within the generic domain, reviewers in the 1994 experiment did slightly better on the PG document than the ATM, while reviewers in the 1995 experiment did slightly worse on the PG document relative to the ATM; while the differences average out when the two runs are combined, the effect still shows up as a significant interaction in the MANOVA test.

Domain	Document	Document * Replication	Document * Group	Document * Replication * Group
Generic	0.7810	0.0298	0.0056	0.5252
NASA	0.9137	0.7672	0.5670	0.5337

Table 2. Effects on individual scores within domains.

Graphs of Least Squares Means are presented in figures 4a and 4b as a convenient way of visualizing the effects of the interaction between document and reading technique. For the generic domain, it can be seen that reviewers in each group on average scored higher with PBR than with the usual technique, taking into account the other effects in the model. In the NASA domain, reviewers in each group scored about the same on both documents, regardless of the technique used. Note that the interaction for the generic domain is significant, providing evidence that reading technique does in fact have an impact on detection rates.

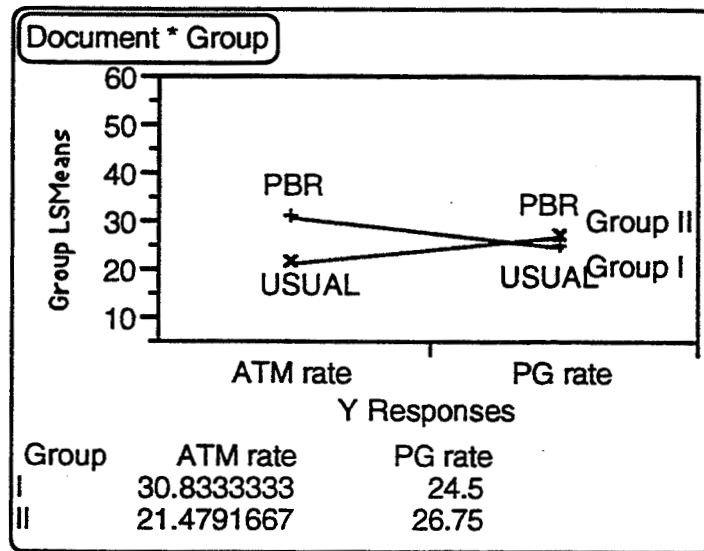


Figure 4a. Interaction between group and technique for the generic domain.

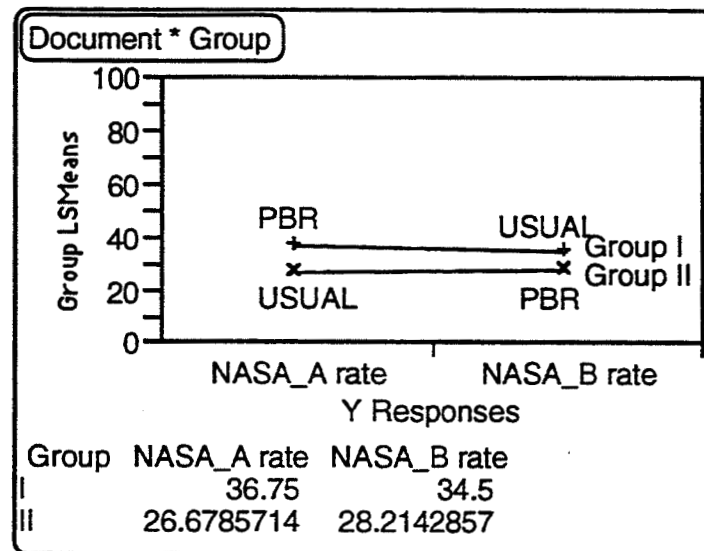


Figure 4b. Interaction between group and technique for the NASA domain.

3.2 Analysis for Teams

3.2.1 Analysis Strategy for Teams

In this section, we return to investigating our primary hypothesis concerning the effect of PBR on inspection teams. The analysis was complicated by the fact that the teams were composed after the experiment's conclusion, and so any grouping of individual reviewers into a team is somewhat arbitrary, and does not signify that the team members actually worked together in any way. The only real constraint on the makeup of a team which applied PBR is that it contain one reviewer using each of the three perspectives; the non-PBR teams can have any three reviewers who applied their usual technique. At the same time, the way in which the teams are composed has a very strong effect on the team scores, so an arbitrary choice can have a significant effect on the test results.

For these reasons, we used a permutation test to test for differences in team scores between the techniques. An informal description of the test follows.

First, since there are differences between the experiment runs, we will compose teams only with reviewers from within the same run; we therefore treat the two experiment runs separately. Results from the individual scores showed that the domains are very different, but the documents within a domain are of comparable difficulty; thus, we compare reviewer scores on documents within the same domain only. We again categorize reviewers into one of two groups, as we did for the analysis within domains for individual scores, depending on which technique they applied to which document. Let us say the reviewers in Group I applied PBR to Document A and their usual technique to Document B, where Document A and Document B represent the two documents within either of the domains. We can then generate all possible PBR teams for Document A and all possible non-PBR teams for Document B, and take the average detection rate of each set. This ensures that our results are independent of any arbitrary choice of team members, but because the data points for all possible teams are not independent (i.e., each reviewer appears multiple times in this list of all possible teams), we cannot run simple statistical tests on these average values. For now, let us call these averages A_I and B_I . We can then perform the same calculations for Group II, in which reviewers applied their usual

technique to Document A and PBR to Document B, in order to obtain averages A_{II} and B_{II} . The test statistic

$$(A_I - B_I) - (A_{II} - B_{II})$$

then gives us some measure of how all possible PBR teams would have performed relative to all possible usual technique teams.

Now suppose we switch a reviewer in Group I with someone from Group II. The new reviewer in Group I will be part of a PBR team for document A even though he used the usual technique on this document, and will be part of a usual technique team for Document B even though he applied PBR. A similar but reversed situation awaits the reviewer who suddenly finds himself in Group II. If the use of PBR does in fact improve team detection scores, one would intuitively expect that as the PBR teams are diluted with usual technique reviewers, the average score will decrease, even as the average score of usual technique teams with more and more PBR members is being raised. Thus, the test statistic computed above will decrease. On the other hand, if PBR does in fact have no effect, then as reviewers are switched between groups the only effect will be due to random effects, and team scores may improve or decrease with no correlation with the reading technique of the reviewers from which they are formed. So, let us now compute the test statistic for all possible permutations of reviewers between Group I and Group II, and rank each of these scenarios in decreasing order by the statistic. If the scenario in which no dilution has occurred appears toward the top of the list (in the top 5%) we will conclude PBR does have a beneficial effect on team scores, since every time the PBR teams were diluted with non-PBR reviewers they tended to perform somewhat worse relative to the usual technique teams. However, should the non-diluted scenario appear toward the middle of the list, then this is clear evidence that every successive dilution had only random effects on team scores, and thus that reading technique is not correlated with team performance.

Note that this is meant to be only a very rough and informal description of the intuition behind the test; the interested reader is referred to Edington's *Randomization Tests* (Edington, 1987).

3.2.2 Results for Teams

The use of the permutation test allows us to formulate and test the following hypotheses:

H₀: The difference between average scores for PBR and usual technique teams is the same for any random assignment of reviewers to groups.

H_a: The difference between average scores for PBR and usual technique teams is significantly higher when the PBR teams are composed of only PBR reviewers and the usual technique teams are composed of only usual technique reviewers.

Level of significance: $\alpha = 0.05$ (that is, we reject H₀ if the undiluted teams appear in the top 5% of all possible permutations between groups)

The results are summarized in Table 3. P-values which are significant at the 0.05-level appear in boldface. For example, twelve reviewers read the generic documents in the 1994 experiment; there are 924 distinct ways they can be assigned into groups of 6. The group in which there was no dilution had the 61st highest test statistic, corresponding to a p-value of 0.0660.

Domain/ Replication	Number of Group Permutations Generated	Rank of Undiluted Group	P-value
Generics/1995	3003	2	0.0007
Generics/1994	924	61	0.0660
NASA/1995	1716	67	0.0390
NASA/1994	924	401	0.4340

Table 3. Results of permutation tests for team scores.

3.3 Analysis for Perspectives

3.3.1 Analysis Strategy for Perspectives

We were also concerned with the question of whether the perspectives used in the experiment are useful (i.e., reviewers using each perspective contributed a significant share of the total defects detected) and orthogonal (i.e., perspectives did not overlap in terms of the set of defects they helped detect). A full study of correlation between the different perspectives and the types and numbers of errors they uncovered will be the subject of future work, but for now we take a qualitative look at the results for each perspective by examining each perspective's coverage of defects and how perspectives overlap.

3.3.2 Results for Perspectives

We formulate no explicit statistical tests concerning the detection rates of reviewers using each of the perspectives, but present Figures 5a and 5b as an illustration of the defect coverage of each perspective. Results within domains are rather similar; therefore we present the ATM coverage charts as an example from the generic domain and the NASA_A charts as an example from the NASA domain. However, due to the differences between experiment runs for the NASA documents, we do not present a coverage diagram for both runs combined. The numbers within each of the circle slices represent the number of defects found by each of the perspectives intersecting there. So, for example, ATM reviewers using the design perspective in the 1995 experiment found 11 defects in total: two were defects that no other perspective caught, three defects were also found by testers, one defect was also found by users, and five defects were found by at least one person from each of the three perspectives.

ATM Results:

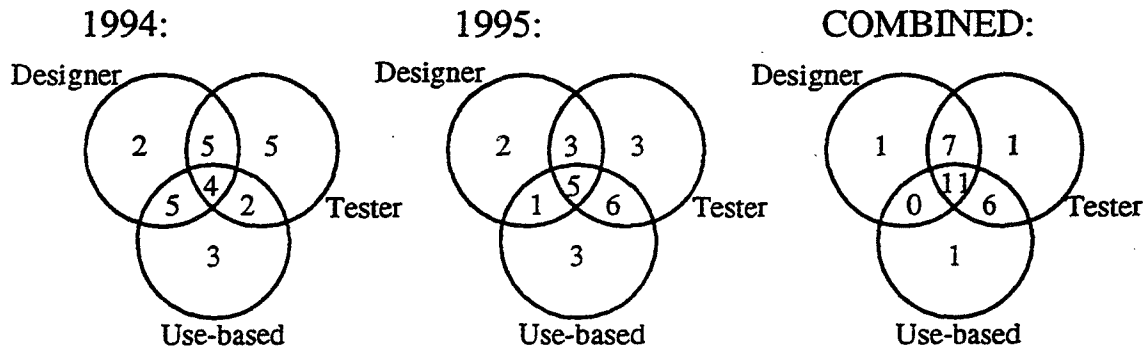


Figure 5a. Defect coverage for the ATM document.

NASA_A Results:

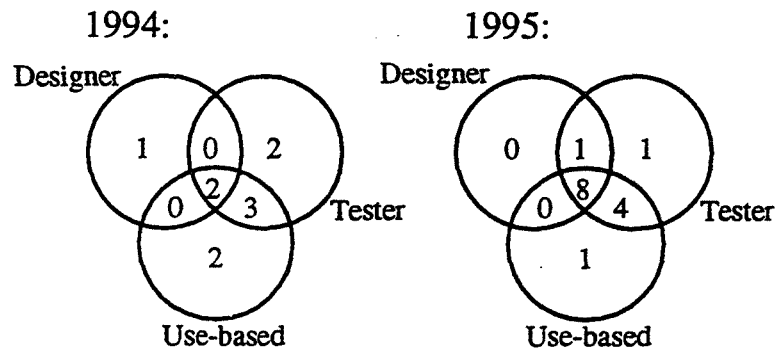


Figure 5b. Defect coverage for the NASA_A document.

4. PBR Effectiveness

In the previous section we presented the analysis of the data from a strictly statistical point of view. However, it is necessary to assess the meaning and implications of the analysis to see if we can identify trends in the results that are similar for both runs of the experiment. Such interpretations may also point out areas of weakness in the experiment or in the PBR technique - weaknesses which upon recognition become potential areas for improvement.

4.1. Individual Effectiveness

4.1.1. The 1994 Experiment

The individual defect detection rates were better for the generic documents than for the NASA documents in the 1994 replication, regardless of reading technique, because the generic documents were simpler to read and less complex than the NASA documents. Most subjects pointed to the size and complexity of the NASA documents as potential problem areas. However, there is a difference not only in absolute score, but also in the impact the technique has on detection rate. The improvement of PBR over the usual technique was greater for the generic documents than for the NASA documents. We can think of various reasons for this:

- The perspectives and the questions provided were not aimed specifically at the NASA documents, but based on the general nature of the generic documents.

Thus the technique itself may not be exploited to its full potential for documents within the NASA domain.

- It is possible that the reviewers are more likely to fall back on their usual technique rather than apply the PBR technique when reading documents that they are familiar with. We received anecdotal evidence of this during follow-up interviews. This may be of particular importance in situations where the subjects are under pressure due to time constraints and the complexity of the document.
- The 1994 experiment was carried out in the reviewers' own work environment. This may increase the temptation to fall back to the usual technique when the familiar situation of reading NASA documents arose. The generic documents, on the other hand, would not be likely to stimulate such interaction effects.
- Insufficient training may have been provided since the training sessions only explained how to use the technique on a sample generic document and not on a sample NASA document.

Within each of the two domains, we found that the documents were at the same level of complexity with only minor differences between them. This indicated that our effort of keeping the documents within each domain comparable was successful.

4.1.2. The 1995 Experiment

In the 1995 replication we made some changes to account for some of the problems mentioned above. The NASA documents were modified substantially according to the comments we received from the subjects. We also provided additional training by adding two more sessions aimed at applying the techniques to the NASA documents. The experiment itself was carried out in a classroom environment instead of the work environment. However, even though we saw a substantial rise in the absolute defect detection rates for the NASA documents, the improvement of PBR over the usual technique remained insignificant. Thus our most viable explanation at the moment is that PBR needs to be more carefully tailored to the specific characteristics of the NASA documents and environment to show an improvement similar to what we see in the generic domain. We also got feedback from the subjects that supported this view; several found it tempting to fall back to their usual technique when reading the NASA documents.

For the generic domain, we made only minor changes to the documents and the seeded defects. Thus, we expected the change in defect detection rate to be negligible. However, this appeared not to be the case.

The mean detection rate for the ATM document turned out to remain unchanged, but dropped significantly for the PG document. We have analyzed this carefully, but have not been able to find a plausible explanation as to why this should happen. Changes to the experiment should be expected to have a similar impact on the two documents, so perhaps the changes to the two documents were not as insignificant as we thought.

4.1.3. Combined

Although the changes to the NASA documents were a definite improvement, any effect due to technique is hidden by the much larger difference between the two runs of the experiment. This problem illustrates one of the tradeoffs we had to make when planning the second run. Should we have kept the documents unchanged, thus getting data that may not be completely valid, or should we change the documents but get data that would be hard to combine with the data from the initial run? We chose to change the documents, and in retrospect we feel the right decision was made.

We did not have the same problems with the generic documents because they were changed only slightly between the two runs of the experiment. Thus the results indicate a significant improvement of the defect detection rate in the generic domain due to the application of PBR.

4.2. Teams

4.2.1. The 1994 Experiment

The defect detection rates of teams in the 1994 experiment reflected the same trends as the individual rates. For the NASA documents, the defect detection rates were much lower than they were for the generic documents, regardless of reading technique. But even more importantly, the results from the permutation test indicate that there are only random differences between the two techniques in this case. This, together with the defect coverage discussed in section 3.2, counts as evidence that the current perspectives do not work as well with the NASA documents as they do with the generic documents.

4.2.2. The 1995 Experiment

In the 1995 experiment, the team results for the generic documents showed that using PBR resulted in a significant improvement over the usual technique. The reasons for this observed improvement, as compared to the 1994 experiment, may include better training sessions and a less intrusive environment, which in the 1995 experiment was a classroom setting. This environment may have made it easier to concentrate on the experiment and thus to keep the two techniques independent from each other.

For the NASA documents, the results were also better than in 1994. In addition to the possible explanations mentioned for the generic documents, there is the fact that there were substantial changes to the documents. Thus, the results provide more evidence for the 1994 indication that the subjects tend to use their usual technique when reading familiar documents in a familiar work environment, and in particular when under pressure.

4.3. Threats to Validity

The threats to internal validity discussed in section 2 may have an impact on the results of the experiment. Thus, at this point it may be interesting to see whether the potential impact and the results agree. Below we discuss the threats that we find most important:

- **History:** One problem with our experiment is that it does not allow history effects to be separated from the change in technique. Since there was one day between the two days of the experiment, some of the improvement that appears due to technique may be attributed to other events that took place between the tests. We do not consider this effect to be very significant, but we cannot completely ignore it.
- **Maturation:** We may assume the results obtained in the afternoon to be worse than the results from the morning session because the subjects may get tired and bored. Since the ordering of documents and domains was different for the two days, the differences between the two days may be disturbed by maturation effects. Looking at the design of the experiment, we see that an improvement from the first to the second day would be amplified for the generic documents, while it would be lessened for the NASA documents. Based on the results from the experiment, we see that this effect seems plausible.

- **Testing:** This may result in an improvement in defect detection rate due to learning the techniques, becoming familiar with the documents, becoming used to the experimental environment and the tests. This effect may amplify the effects of the historical events and thus be part of the reason for improvement that has previously been considered a result of change in technique. Testing effects may counteract maturation effects within each day.
- **Reactive effects:** The change of experimental environment between the experiment runs may have made it easier to concentrate on the techniques and tests to be done, thus separating the techniques better for the second run of the experiment.

We cannot say anything conclusive about the impact of threats to validity. However, we feel that we have taken them into account as carefully as possible, given the nature of the problem and our experimental design.

Since the two runs of this experiment have been done in close cooperation with the NASA SEL environment, it seems natural to conclude this section with a discussion of the extent to which the results can be generalized to a NASA SEL context. This kind of generalization involves less of a change in context than is the case for an arbitrary organization; in particular the differences in populations can be ignored since the population for the experiments is in fact all of the NASA SEL developers.

Clearly, the results for the generic documents cannot be generalized to the NASA documents due to the difference in nature between the two sets of documents. The results for the NASA documents, on the other hand, may be valid since we used parts of *real* NASA documents. Finally, there is a potential threat to validity in the choice of experimental environment. In 1994, the experiment was carried out in the subjects' own environment, and thus would be valid also in a real setting. We cannot assume the same for the 1995 results since this run was done in a classroom situation.

5. Observations on Experimental Design

We have encountered problems in the two runs of the experiment which we have previously discussed. However, some of these problems are of a general nature and may be relevant in other experimental situations.

- *What is a good design for the experiment under investigation, given the constraints?*

There appears to be no easy answer to this question. Each design will be a result of a number of tradeoffs, and it is not always possible to know how the decisions will influence the data. A good design can have various interpretations based on what are considered the goals for the experiment. One option is to use different designs involving different threats to validity and study the results as a whole.

- *What is the optimal sample size? Small samples lead to problems in the statistical analysis while large samples represent major expenses for the organization providing the subjects.*

Organizations generally have limits for the amount of subjects they are willing to part with for an experiment, so the cost concerns are handled by the organizations themselves. A small sample size requires us to be careful in the design in order to get as many useful data points as possible. For this experiment, an example of such a tradeoff is that we chose to neglect learning effects in order to avoid spending subjects on control groups. This gave us more data points to be used in analyzing the difference between the two techniques, but at the same time we remained uncertain as far as the threat to internal validity caused by learning effects is concerned.

- *We need to adjust to various constraints - how far can we go before the value of the experiment decreases to a level where it is not worthwhile?*

Our problem as experimenters is to maintain a certain level of validity while still generating sufficient interest for an organization to allow us to conduct the experiment. From an organization's point of view, an experiment should be closely tied to their own environment to see if the suggested improvement works with minimal effort in terms of environmental changes. From an experimental point of view, however, we are interested in a controlled environment where disturbing interaction effects are negligible.

- *To what extent can experimental aspects such as design, instrumentation and environment be changed when the experiment still is to be considered a replication?*

One requirement for being considered a replication is that the main hypotheses are the same. Changes in design and instrumentation, in particular to overcome threats to

validity, should also be considered "legal". However, one situation we should avoid is making substantial changes to the design based on the *results* from a previous experiment. This will introduce dependencies between the experiments that are highly undesirable from a statistical point of view.

For this experiment in particular, there are various problems that we need to study more carefully. The threats to validity should be carefully examined; in particular we feel the testing effects to be crucial. An experiment with a control group could be one way of estimating what the importance of these effects really are. We may also consider a more careful analysis of the NASA documents and environment in order to refine PBR to these particular needs. The results indicate that the choice of perspectives and associated scenarios do not match the needs of the NASA domain.

A more fundamental problem that should be considered is to what extent the proposed technique actually is followed. This problem with process conformance is relevant in experiments, but also in software development where deviations from the process to be followed may lead to wrong interpretation of measures obtained. For experiments, one problem is that the mere action of controlling or measuring conformance may have an impact on how well the techniques work, thus decreasing the external validity.

Conformance is relevant in this experiment because there seems to be a difference that corresponds to experience level. Subjects with less experience seem to follow PBR more closely ("It really helps to have a perspective because it focuses my questions. I get confused trying to wear all the hats!"), while people with more experience were more likely to fall back to their usual technique ("I reverted to what I normally do.").

There are numerous alternative directions for the continuation of this research. For further experimentation within NASA's SEL it seems to be necessary to tailor PBR to more closely match the particular needs of that domain. A possible way of further experimentation would be to do a case-study of a NASA SEL project to obtain more qualitative data.

We may also consider replication of the generic part of the experiment in other environments, perhaps even in other countries where differences in language and culture may cause effects that can be interesting targets for further investigation. These replications can take the form of controlled experiments with students, controlled experiments with

subjects from the industry using their usual technique for comparison, or case studies in industrial projects.

One challenging goal of a continued series of experiments will be to assess the impact that the threats to validity have. Since it is often hard to design the experiment in a way that controls for most of the threats, a possibility would be to concentrate on certain threats in each replication to assess their impact on the results. For example, one replication may use control groups to measure the effect of repeated tests, while another replication may test explicitly for maturation effects. However, we need to keep the replications under control as far as threats to *external* validity are concerned, since we need to assume that the effects we observe in one replication will also occur in the others.

Acknowledgements

This research was sponsored in part by grant NSG-5123 from NASA Goddard Space Flight Center to the University of Maryland. We would also like to thank the members of the Experimental Software Engineering Group at the University of Maryland for their valuable comments to this paper.

References

- (Campbell, 1963) Campbell, Donald T. and Stanley, Julian C. 1963. *Experimental and Quasi-Experimental Designs for Research*. Boston, MA: Houghton Mifflin Company.
- (Edington 1987) Edington, Eugene S. 1987. *Randomization Tests*. New York, NY: Marcel Dekker Inc.
- (Fagan, 1976) Fagan, M. E. 1976. *Design and code inspections to reduce errors in program development*. IBM Systems Journal, 15(3):182-211.
- (Hatcher, 1994) Hatcher, Larry and Stepanski, Edward J. 1994. *A Step-by-Step Approach to Using the SAS® System for Univariate and Multivariate Statistics*. Cary, NC: SAS Institute Inc.²

- (Heninger, 1980) Heninger, Kathryn L. 1985 *Specifying Software Requirements for Complex Systems: New Techniques and Their Application*. IEEE Transaction on Software Engineering, SE-6(1):2-13
- (Linger, 1979) Linger, R. C., Mills H. D. and Witt, B. I. 1979. *Structured Programming: Theory and Practice*. In The Systems Programming Series. Addison Wesley.
- (Parnas, 1985) Parnas, Dave L. and Weiss, David M. 1985. *Active design reviews: principles and practices*. In Proceedings of the 8th International Conference on Software Engineering, p.215-222.
- (Porter, 1995) Porter, Adam A., Votta, Lawrence G. Jr. and Basili, Victor R. *Comparing Detection Methods For Software Requirements Inspections: A Replicated Experiment*. IEEE Transactions on Software Engineering, June 1995.
- (SAS, 1989) SAS Institute Inc. 1989. *JMP® User's Guide*. Cary, NC: SAS Institute Inc.³
- (SEL, 1992) Software Engineering Laboratory Series. 1992. *Recommended Approach to Software Development, Revision 3*, SEL-81-305, p. 41-62.
- (Votta, 1993) Votta, Lawrence G. Jr. 1993 *Does every inspection need a meeting?* In Proceedings of ACM SIGSOFT '93 Symposium on Foundations of Software Engineering. Association of Computing Machinery, December 1993.

A. Sample Requirements

Below is a sample requirement from the ATM document which tells what is expected when the bank computer gets a request from the ATM to verify an account:

Functional requirement 1

Description: The bank computer checks if the bank code is valid. A bank code is valid if the cash card was issued by the bank.

Input: Request from the ATM to verify card (Serial number and password)

Processing: Check if the cash card was issued by the bank.

Output: Valid or invalid bank code.

We also include a sample requirement from one of the NASA documents in order to give a picture of the difference in nature between the two domains. Below is the process step for calculating adjusted measurement times:

Calculate Adjusted Measurement Times: Process

1. Compute the adjusted Sun angle time from the new packet by

$$t_{s,adj} = t_s + t_{s,bias}$$

2. Compute the adjusted MTA measurement time from the new packet by

$$t_{T,adj} = t_T + t_{T,bias}$$

3. Compute the adjusted nadir angle time from the new packet.

- a. Select the most recent Earth_in crossing time that occurs before the Earth_in crossing time of the new packet. Note that the Earth_in crossing time may be from a previous packet. Check that the times are part of the same spin period by

$$t_{e-in} - t_{e-out} < E_{max} T_{spin,user}$$

- b. If the Earth_in and Earth_out crossing times are part of the same spin period, compute the adjusted nadir angle time by

$$t_{e-adj} = \frac{t_{e-in} + t_{e-out}}{2} + t_{e,bias}$$

4. Add the new packet adjusted times, measurements, and quality flags into the first buffer position, shifting the remainder of the buffer appropriately.

5. The Nth buffer position indicates the current measurements, observation times, and quality flags, to be used in the remaining Adjust Processed Data section. If the Nth buffer does not contain all of the adjusted times ($t_{s,adj}$, $t_{b,adj}$, $t_{T,adj}$, and $t_{e,adj}$), set the corresponding time quality flags to indicate invalid data.

Footnotes

¹ ISERN is the International Software Engineering Research Network whose goal is to support experimental research and the replication of experiments.

² SAS® is the registered trademark of SAS Institute Inc.

³ JMP® is a trademark of SAS Institute Inc.

59-61
415475
CLOSE
360857
201

The Impact of Ada and Object-Oriented Design in NASA Goddard's Flight Dynamics Division

Sharon Waligora
Computer Sciences Corporation
10110 Aerospace Rd.
Lanham-Seabrook, MD 20706
301-794-1744
swaligor@csc.com

John Bailey
Software Metrics, Inc.
4345 High Ridge Rd.
Haymarket, VA 22069
703-855-4472

Mike Stark
NASA/Goddard
Software Engineering Branch
Greenbelt, MD 20771
301-286-5048
michael.e.stark@gsfc.nasa.gov

Abstract

This paper presents the highlights and key findings of 10 years of use and study of Ada and object-oriented design in NASA Goddard's Flight Dynamics Division (FDD). In 1985, the Software Engineering Laboratory (SEL) began investigating how the Ada language might apply to FDD software development projects. Although they began cautiously using Ada on only a few pilot projects, they expected that, if the Ada pilots showed promising results, the FDD would fully transition its entire development organization from FORTRAN to Ada within 10 years. However, 10 years later, the FDD still produced 80 percent of its software in FORTRAN and had begun using C and C++, despite positive results on Ada projects. This paper presents the final results of a SEL study to quantify the impact of Ada in the FDD, to determine why Ada has not flourished, and to recommend future directions regarding Ada. Project trends in both languages are examined as are external factors and cultural issues that affected the infusion of this technology. The detailed results of this study were published in a formal study report [1] in March of 1995. This paper supersedes the preliminary results of this study that were presented at the Eighteenth Annual Software Engineering Workshop in 1993 [2].

Introduction

Beginning in 1985, the Flight Dynamics Division (FDD) at NASA's Goddard Space Flight Center and Computer Sciences Corporation (its primary support contractor) began investigating Ada and object-oriented design (OOD) as means of improving its products and reducing development costs. The FDD's intention was to become an Ada development "shop" within 10 years. This decision was based on widespread opinion in the software engineering community, particularly among U.S. Government agencies, that Ada was "more than just another programming language." It was felt that Ada represented a significant advance in software engineering technology that would lead to better products and a more disciplined practice of software engineering. Ada had been designed by the Department of Defense (DoD) with the goal of providing a common language that would support the portability of programs, tools, and personnel across many projects. Another goal was to provide, in Ada, a

tool beneficial for large-system development and long-term maintenance.

The Software Engineering Laboratory (SEL), which facilitates software process improvement within the FDD through an organized measurement, research, and technology infusion program, selected Ada as one of several software engineering technologies available at that time that had potential for significantly improving the local software process and products. During its initial experimentation with the language, the SEL chose to combine Ada with OOD to extend its impact throughout the full software development life cycle and to ensure that new design approaches would be explored.*

*Strictly speaking, Ada 83 is not an object-oriented language, because it does not support dynamic binding and inheritance. However, it does support objects, and OOD methodologies can take full advantage of Ada. Ada 95 more fully supports the object-oriented paradigm.

The FDD's investigation of Ada/OOD was conducted as a series of experimental projects and deliverable Ada systems. Ada experiments and projects were assigned specific goals addressing different aspects of software development, such as design concepts, software reuse, cost and schedule adherence, and system performance. Progress toward these goals was guided and monitored by the SEL and documented in study reports summarizing the results and lessons learned from each of the Ada experiences. Project characteristic data for the Ada systems were collected and stored in the SEL data base along with data for earlier FDD projects (before 1985) and concurrent FORTRAN projects (1985-1994).

This paper summarizes the results of an in-depth investigation into the Ada experience in this organization, commissioned by the FDD and the SEL. Conducted in association with an outside consultant (Software Metrics, Inc.), this investigation gathered together the sum of the research and experience described above; quantitative data (system size, effort, errors, project duration, percent reuse, and performance) for all projects, both FORTRAN and Ada, active between 1985-1994; and the opinions of FDD personnel, both those directly involved in the transition and those simply present in the environment during the period. These materials have been analyzed with a focus on the evolution of local products and processes since Ada/OOD have been in use. Significant improvements in product characteristics have been documented, as well as notable changes to the software development process. This investigation also sought and identified the reasons why, 10 years after introducing this technology with an expressed intention of fully transitioning to it, and after witnessing improvements in product and process, the FDD develops only 15-20% of its software using Ada. Complete and detailed findings of this investigation are presented in the study team's final report, *Impact of Ada and Object-Oriented Design in the Flight Dynamics Division at Goddard Space Flight Center* [1] (available via the Word Wide Web).

Because the FDD typically uses a single language to develop small to mid-sized systems with relatively short life spans, this organization was not able to apply Ada in the context for which it was originally designed. Hence, readers should bear in mind that this study reports only one experience with this technology. As the findings suggest, the language offers clear benefits and involves significant investment. The specific influential factors in any one organization (e.g., software domain, hardware environment, long-term

goals) must be considered in any evaluation of Ada's applicability and effectiveness.

Experience With Ada in the FDD

The FDD's activities during its transition to Ada fall into three categories that span overlapping phases throughout the study period: experimentation and study, pilot operational use, and routine operational use in one small application domain. Figure 1 shows the timeline for the various projects, research efforts, and studies conducted and the activity or transition phase in which they are included.

Experimentation and study have continued throughout the transition period, providing a context for investigating new approaches and resolving critical issues. The Ada work began with an experiment in 1985 that was designed to foster learning about the language and its applicability in the FDD while posing minimal risk to the operational environment. This initial experiment was conducted as a parallel effort, where two versions of the same system were developed: one in FORTRAN (operational version) and another in Ada (study version). Following this experiment, the FDD developed a series of research prototypes to investigate new ways of using Ada that would lead to future advancements (e.g., reconfigurable software). Additionally, the SEL conducted several studies to probe more deeply into issues raised by pilot projects (e.g., performance) and to better understand areas that the pilot projects would not encounter (e.g., portability).

Pilot operational use began in 1987, when the FDD began using Ada to develop small, low risk operational simulators. Each of these pilot projects focused on specific goals and contributed to the evolution of the use of Ada in the FDD, in addition to producing software systems that were used for actual mission support. Finally in 1990, the FDD began to use Ada routinely on one class of software systems, telemetry simulators.

Goals and Expectations

The overall goal of the FDD was to reduce development cost and cycle time for producing flight dynamics mission support systems by maximizing reuse. Ada and OOD had potential for significantly increasing reusability. In addition, the FDD was interested in adopting the high-quality software engineering practices supported and encouraged by these technologies. As local experience with Ada/

OOD grew, specific subgoals evolved within the context of the overall goal, which helped establish areas of focus for individual projects and studies. The proliferation of languages (the DoD's original concern), however, was not an issue because the FDD had always used only one or two languages for its

system development.

Over this 10-year period, the FDD has delivered approximately 1 million lines of Ada code. Figure 2 illustrates the growth of Ada experience in this environment. The curve shows the accumulated amount of code (in thousands of lines of code (KSLOC)) as

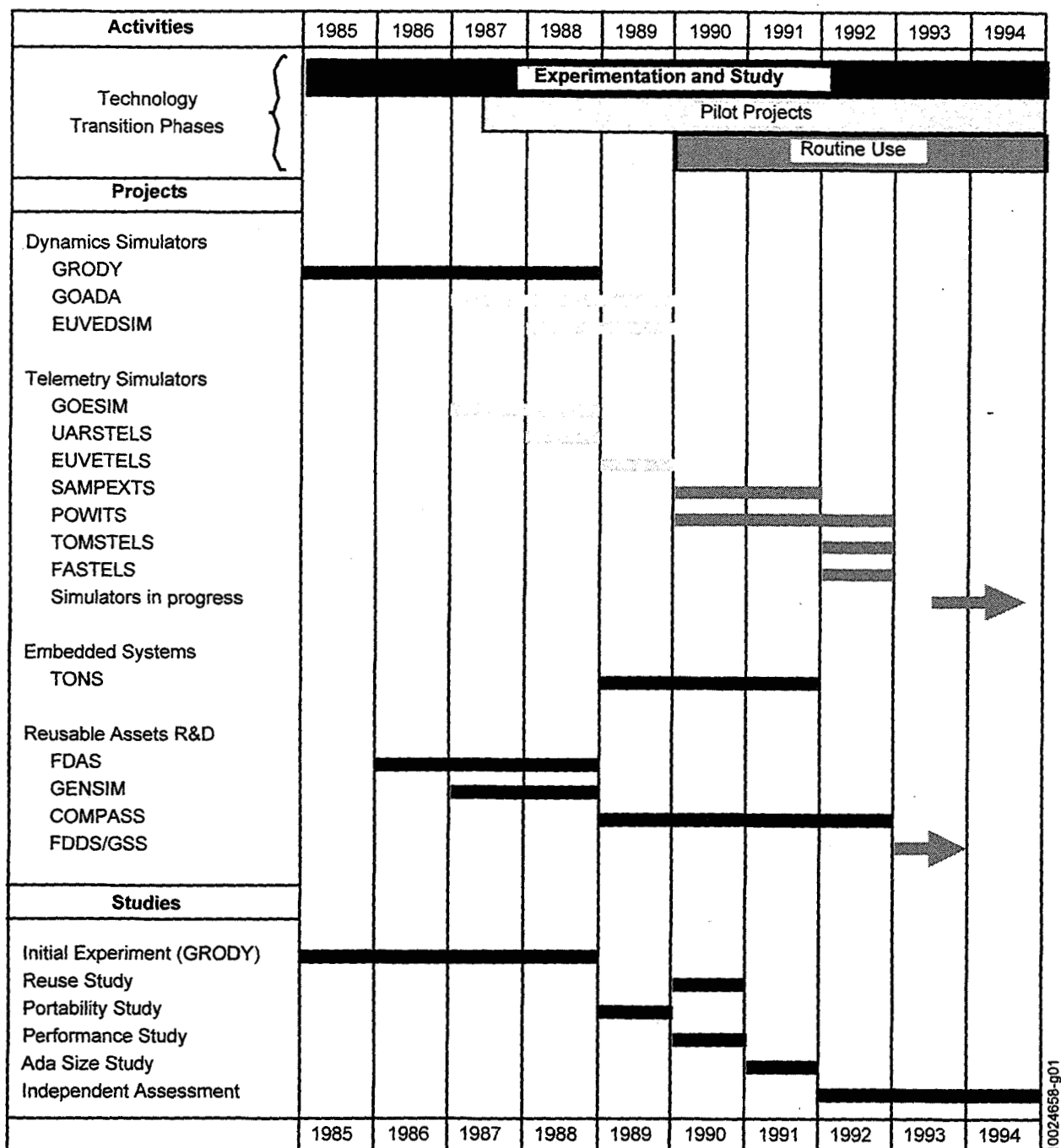


Figure 1. FDD Ada Activity Timeline

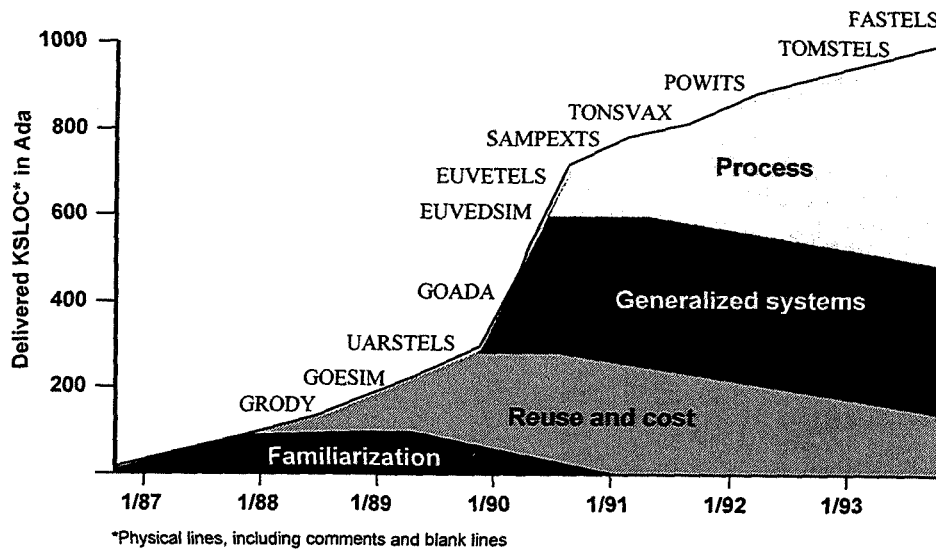


Figure 2. FDD Ada Goals and Experience

each project was delivered (the time before the first project delivery is foreshortened for clarity).

The four regions under the curve in Figure 2 give a rough approximation of the evolution of goals and objectives for the study and use of Ada in the FDD. Initially, the main concern was familiarization with the language, although the initial projects also stressed reusability as a primary objective. Soon, the focus turned to the structured generalization of systems, and the success of these generalizations led to an overall improvement in the efficiency of the Ada software development process. Recently, there has been an additional focus on optimizing the development process specifically for use with the Ada language. This optimized process has been specified and documented in a recent supplement [3] to the standard software development process guidebook used by the FDD [4]. These Ada study goals for reuse, generalization, and process provided the framework for the evolving use of Ada in this environment.

Each of the Ada projects and studies furthered the FDD's understanding of Ada and the organization's progress toward its overall goal. As short-term goals guided and focused the projects, project experience and study findings adjusted the FDD's course as it transitioned to Ada. Reference 1 provides a time-ordered description of the Ada project goals and key experiences.

Quantitative Analysis

The success with which the FDD met its Ada experimentation goals of increased software reuse,

lower development effort, shorter cycle times, and greater software reliability was evaluated by analyzing data from contemporaneous Ada and FORTRAN flight dynamics projects [5]. Previous papers have documented improvements achieved on Ada projects over the 1985 FORTRAN baseline [6]. But, while the FDD was gradually maturing its use of Ada for satellite simulators built on DEC VAX minicomputers, the FORTRAN process used on the larger, mainframe-based projects was also evolving and improving.

This section compares the evolving Ada and FORTRAN baselines between 1985 and 1994 in each of the four experimentation goal areas (reuse, cost, schedule, and reliability) and discusses the evolving software process. It also presents a summary of the results of quantitative analyses of data on process and performance. Any improvements seen on the Ada projects are assessed within the context of the evolving FORTRAN baseline. Since the preliminary SEL report on this study [2], new data have been added for completed projects in both languages, and the size and effort data have been normalized to support a more accurate comparison among projects in the two languages.

Project Data

The FDD delivered operational software to support 10 spacecraft missions from 1985 to 1994. Of these, eight missions had at least one simulator built in Ada on the VAX and an attitude ground support system developed in FORTRAN on the IBM mainframe computer. Data from all FDD projects that produced operational

software for these eight missions were examined. In particular, the series of corresponding telemetry simulators and ground support systems from the same missions were analyzed to assess the relative impact of using Ada and FORTRAN.

For each language, projects were grouped according to date (1985–1989 and 1990–1994), producing two distinct analysis periods. This division into “early” and “recent” projects occurs at a natural break in the data that corresponds to a significant increase in levels of reuse achieved and to changes in the local development process.

Software size is used in these analyses as a normalizing factor when comparing productivity, reuse, error density, and process effects. The traditional measure of software size in the FDD has been source lines of code (SLOC), which counts every carriage return in the source files, including blank lines and comments. For this study, however, statement counts were chosen (i.e., the number of logical statements and declarations). The statement count is not sensitive to formatting and therefore provides a more uniform indicator across the two languages of delivered functionality and of development effort expended [7].

Reuse

During the time that Ada has been used in the FDD, the reuse of previously developed software on new projects has increased considerably. This has been achieved on all FDD projects that have applied object-oriented methods, regardless of language. Figure 3 and Figure 4 show, for Ada and FORTRAN projects, respectively, the percentage of each project that was reused without change (verbatim) from previous projects. The minimum unit of reuse is a single compilation unit; no credit is given if only a portion of a compilation unit is reused. The percentages are computed by dividing the total size of all compilation units reused verbatim by the total delivered size of the project.

Figure 3 shows a large increase in verbatim reuse that occurred in 1989 when a set of Ada generics purposely designed for reuse during the UARSTELS project was demonstrated to be sufficient to construct nearly 90% of EUVETELS, the subsequent project in the telemetry simulator domain. This level was maintained for telemetry simulators until the POWITS project (dip in the amount of reuse shown in Figure 3), when a change in the domain required that the Ada generics be modified and additional new code be developed. Specifically, the original domain where high reuse was

achieved was simulation software for three-axis-stabilized spacecraft. When a spin-axis-stabilized spacecraft was simulated for the first time, a substantial drop occurred in the verbatim reusability of the library generics. This incompatibility was rectified with the creation of additional generics so that now the entire set can accommodate either a three-axis- or a spin-axis-stabilized spacecraft. The slight drop in the most recent examples of reuse to around 70%, compared with the earlier successes that were closer to 90%, was due to performance tuning on the latest projects.

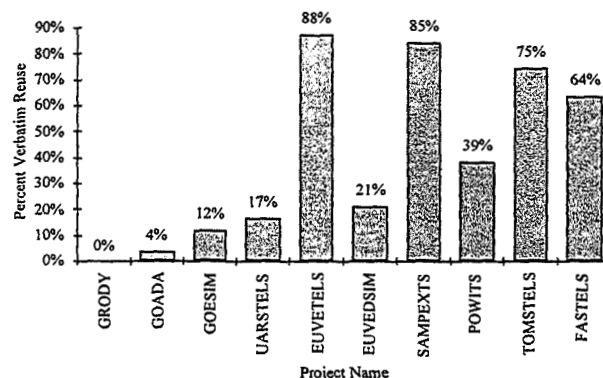


Figure 3. Verbatim Reuse Percentages for Ada Projects

Figure 4 shows the corresponding picture of verbatim reuse on the FORTRAN projects during the same period. At its peak, the amount of verbatim reuse achieved was nearly as great as with the reusable Ada generics, and the first successes occurred at nearly the same time as the first highly successful Ada reuse. (The first high-reuse FORTRAN project, EUVEAGSS, was the corresponding ground support system for the same satellite mission as the first high-reuse Ada simulator, EUVETELS.) Again, the change in domain to spin-stabilized spacecraft caused a drop back to the low levels of reuse observed on the earlier projects in the late 1980s. In the FORTRAN case, however, the reusable components from the three-axis domain were even less suited to the spin-stabilized domain than had been the case with the Ada components. This is shown by the even greater drop in reuse on the FORTRAN ISTP system compared with the corresponding Ada simulator, POWITS.

Before drawing conclusions from these data, it is important to understand the different reuse approaches that have been used on FORTRAN and Ada projects and to consider their effect on quantitative data. The independent assessment team thoroughly investigated the different reuse approaches to determine their influence on the quantitative results.

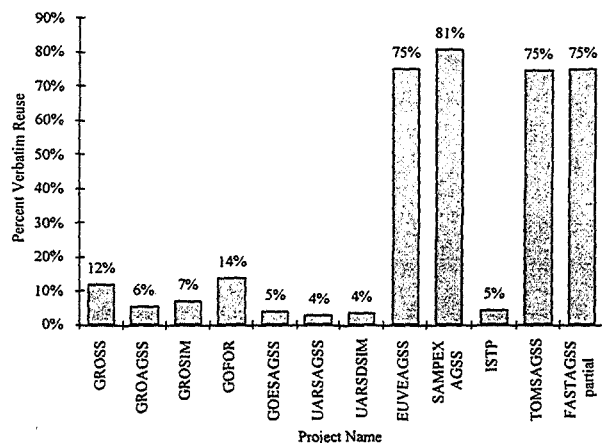


Figure 4. Verbatim Reuse Percentages for FORTRAN Projects

Different Reuse Methods

The FDD used two different methods to manage reuse on its Ada and FORTRAN projects. This decision had more to do with the amount and expected lifetime of the software being reused, than it did with language. The ground systems are very large and are used for many years to support active spacecraft missions; this makes strict, controlled management of the reusable

code common to all ground systems very important. Thus, the FDD chose to create a central library containing the reusable FORTRAN ground system subsystems and to allocate a separate library maintenance team to both maintain it for all active missions and modify it to support all new projects. Because of this, individual ground system project teams are only responsible for developing new mission-specific subsystems, which they execute in concert with selected standard reusable subsystems to support a mission. Conversely, the Ada simulators are relatively small with short operational lifetimes (on the order of months—to support prelaunch testing); this makes long-term configuration management a less important concern. Thus, Ada projects employing reusable components maintain and modify their own copy of the reusable simulator software for each mission.

Table 1 summarizes the basic differences between the reuse methods used on the FORTRAN and Ada projects. It is important to consider these differences when analyzing the quantitative data in order to differentiate, as much as is possible, between the effects of the reuse methods and the effects of using different languages.

Table 1. Ada vs. FORTRAN Reuse Methods

Factor	FORTRAN Systems	Ada Systems
Reusable source code management approach	Single library serves all development projects and operational missions.	Each development project and operational mission has its own copy of the reusable source code.
Generalization approach for implementing reusable software	Package data and functionality together. Use case statements to handle multiple mission needs. (Not all reusable software used object-oriented design.)	Use Ada generic packages to implement parameterized logic that is instantiated for specific mission at compile time via parameters.
Reuse approach	New mission-specific subsystems communicate with reused executables via data sets at run-time.	New and modified units are linked with verbatim reused units to produce project executable.
Personnel	Separate, specialized team maintains (modifies and tests) reusable code to fit new mission requirements. Project team develops new mission-specific subsystems.	Project team modifies reusable code when necessary and develops new mission-specific components.
Change philosophy	New mission requirements that affect reusable subsystems are handled by appending mission-specific 'case' logic to generalized subsystems; existing code is not touched if possible. Rigorous regression testing is done.	Mission-specific requirements are handled through parameterized generics. When modifications are necessary, the generic components are made more generalized to handle the new requirements also.

Software Size Differences Due to Generalization Approach and Language

The verbatim reuse percentages reported in the project data give the impression that the two languages are equally able to express generalized functionality. However, further investigation revealed significant differences in the structure of the reusable code and the software change philosophy used depending on the language used.

The Ada reusable architecture makes extensive use of Ada generics to provide generalized packages and procedures, which are instantiated to create mission-specific code during compilation. Because some of the generics are used repeatedly within the reusable software, the net effect was a 25% decrease (for a compression factor of .75) in the amount of code required to implement equivalent functionality when compared with earlier single-purpose mission-specific Ada code. The FORTRAN reusable subsystems similarly used object-oriented techniques to encapsulate data and functionality into reusable components; however, the generalization was provided using parameterized case statements to determine (at run-time) which code to execute for a particular mission. Specific code was provided for each individual case. The FORTRAN type of generalization caused the reusable code size to grow. For example, an analysis of the generalized subsystems showed that they increased in size between 10% and 40% when compared with subsystems expressing similar functionality in earlier mission-specific systems; this indicates an expansion average of about 25% (or a factor of 1.25).

The maintenance approach for the reusable software, which is driven both by language and generalization approach, also affects software size. FORTRAN libraries must be continually augmented to handle new missions in their respective domains. It is the practice of the FORTRAN maintainers to augment the subsystems as necessary by adding code for any new requirements rather than by generalizing or modifying the existing code. This approach is more straightforward given the limitations of FORTRAN, and it also avoids the risk of introducing errors for existing clients. However, this also causes the FORTRAN libraries to grow over time. For example, the generalized subsystems used in both the EUVEAGSS and the TOMSAGSS have grown by nearly 10% while under maintenance. Conversely, the Ada generics form a set of smaller components that requires little or no further modification to handle missions in either domain. The Ada developers directly

handle the generics needed for each project and further generalize them only when necessary (such as by deleting unnecessary dependencies between components). Thus the size of the reusable Ada software remains roughly constant.

These size differences have the following implications:

- The generalized FORTRAN systems are on the average 25% larger than previous systems that provide similar functionality. This was considered when productivity measures were examined for this analysis. Combining the FORTRAN expansion ratio of 1.25:1 and the Ada compression factor of .75:1 results in a net difference of 1.5:1 between FORTRAN and Ada generalized software size.
- The large, and continually growing, size of the FORTRAN reusable library increases the cost of maintaining it. While 3500 hours were required to enhance the generalized subsystems for SAMPEX (in 1991–1992), it cost between 5000 and 6500 hours per mission to enhance the same generalized software for use by four mission systems in late 1992 through 1994.

Process Evolution

An evolving development process had as much to do with the improvements in productivity as did the increases in software reuse. Without the support of an appropriate process, reuse techniques alone would not have led to the improvements observed. The software process is characterized by examining the distribution of effort across the various software development activities performed. Life-cycle activity categories include design, code, test, and "other" (e.g., management, meetings, system documentation).

The differing shapes of the early and recent activity distributions shown in Figure 5 illustrate that the more recent, high-reuse Ada projects have been, in fact, conducted using a different process than the early projects. The light bars for each activity show the averages for the first five Ada simulator projects, and the dark bars show the average effort per activity for the five recent Ada simulators that achieved higher levels of reuse. In both cases, the percentages are based on the average total effort for projects in the early set to more dramatically demonstrate the savings realized on recent projects relative to those earlier projects.

Because savings were apparent in each of the activities, it was concluded that the savings exhibited for the

recent project set are not due to code reuse alone but also to the process change that came about as a by-product of that reuse. Some of the process changes include requirements specifications expressed in terms of specific earlier system functionality, compression of preliminary and critical design reviews into one review, and reuse of baseline documentation.

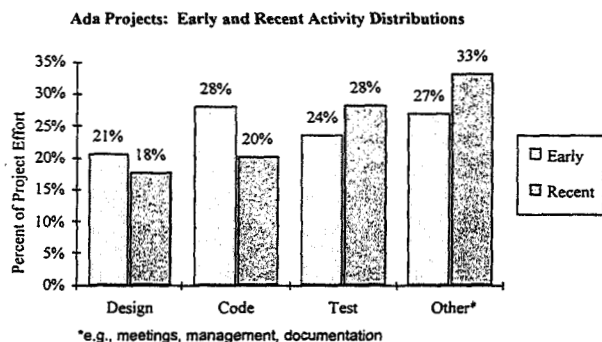


Figure 5. Activity Distribution for Ada Projects

Figure 6 shows the average effort by activity for the FORTRAN projects that were completed during the same period. Again, the projects are divided into an earlier group of lower reuse projects and a more recent group of higher reuse projects, and the percentages are all based on the average early project effort. As with Ada, a reduction in effort is shown for each activity when comparing low reuse with high reuse, although the net reduction is less in FORTRAN. Unlike the Ada results, however, most of the FORTRAN reduction occurs in the coding activity instead of being spread more evenly across the life cycle.

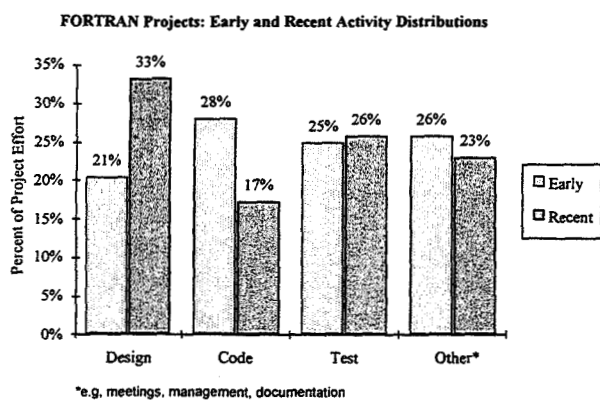


Figure 6. Activity Distribution for FORTRAN Projects

The shape of the activity distribution of the early projects in the FORTRAN set (light bars in Figure 6) is virtually identical to the activity distribution of the

early projects in the Ada set (light bars in Figure 5). On the other hand, the distributions for the recent high-reuse projects differ between the languages. This suggests that the Ada and FORTRAN processes have evolved in different ways even though they share a common ancestry. The main lessons from this illustration are that the software process matured and improved during the Ada exploration period and that this evolution affected both the Ada work and the FORTRAN work, although in different ways. Discussions later in this section will show that these process changes are also associated with a reduction in overall project cost and shortening of schedules.

Cost Reduction

The average cost to deliver a statement in each language was calculated, again adopting the distinction between conventional-reuse projects and high-reuse projects—respectively, those before and after the EUVETELS project. The left-hand side of Figure 7 shows the average cost in hours to deliver a statement of Ada, both for the early project set and the recent project set. The figure shows that the net productivity of delivering Ada software has doubled since high reuse has been achieved.

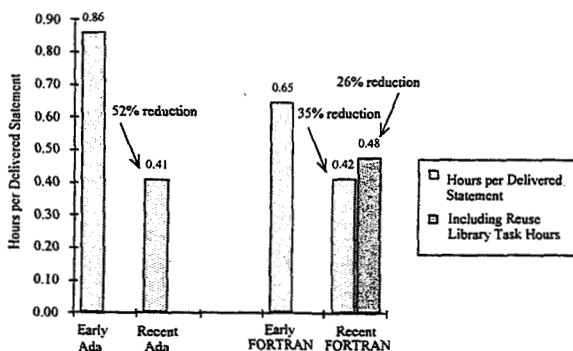


Figure 7. Average Effort to Deliver a Statement

The right-hand side of the figure shows the average cost in hours to deliver a statement of FORTRAN before and after the high-reuse process. Again, there is an improvement, though not as great a reduction as in the Ada projects, particularly when the effort of the library maintenance team is computed in the total.

The change in the cost shown to deliver the most recent FORTRAN projects reveals that, due to the overhead of maintaining the reuse libraries, there has been less net improvement in the efficiency of FORTRAN development since adopting the high-reuse process. This suggests that, although FORTRAN is probably

more cost effective for building short-lived, single-use software, Ada is preferable for software that is likely to have a longer life through future reuse.

The effect on code size when expressing general software in each language also must be taken into consideration when using statement counts to compare productivity. As discussed earlier, the generalized portions of the FORTRAN code were larger than the comparable single-purpose solutions. Conversely, the Ada projects that incorporated the reusable generics were somewhat smaller than the earlier similar projects. The net difference between the two languages, approximately 1.5 to 1, means that the effective cost for Ada (based on functionality delivered) is actually lower than that shown above, whereas the effective cost for FORTRAN is actually higher. Adjusting for the Ada size compression factor of .75:1 and the corresponding size expansion factor of 1.25:1 for the generalized parts of the FORTRAN results in a more accurate picture of the change in cost due to high reuse in each language. Figure 8 shows the cost of delivering comparable functionality between the early project set (low reuse) and the recent project set (high reuse) for both Ada and FORTRAN. This indicates that, in terms of functionality, FORTRAN development costs have decreased only slightly, whereas Ada costs have come down by nearly two-thirds due to high reuse.

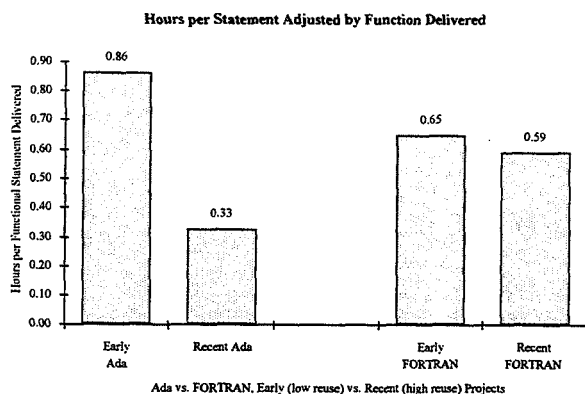


Figure 8. Average Effort To Deliver Similar Functionality

Schedule Compression

In addition to lowering cost, Ada and reuse were also expected to lead to shorter cycle times or project durations. Figure 9 shows that this goal was met not only by the Ada projects but also by the FORTRAN projects. Again, the right-hand bars in each pair represent the "recent" projects, or those that achieved high reuse levels. Because the FORTRAN ground

systems and the Ada telemetry simulators are affected by different external forces, a cross-language comparison of cycle time makes no sense. But a comparison of the early and recent project groups in each language shows improvement.

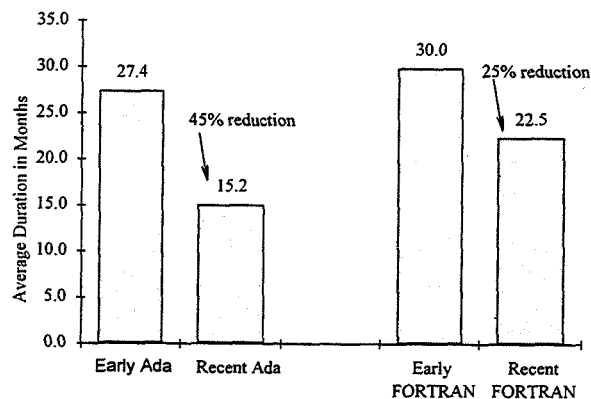


Figure 9. Average Project Duration

Reliability

The last explicit goal for the planned Ada transition was to increase the quality of the delivered systems. The density of errors discovered during development, which is measured on all FDD projects, was used to represent system quality and reliability (there was insufficient operational data to conduct a reliability analysis). Development-time errors are a useful indication of quality because they reveal the potential for latent undetected errors and indicate spoilage and rework during development. Figure 10 shows the number of errors discovered per thousand statements of new and modified software before delivery.

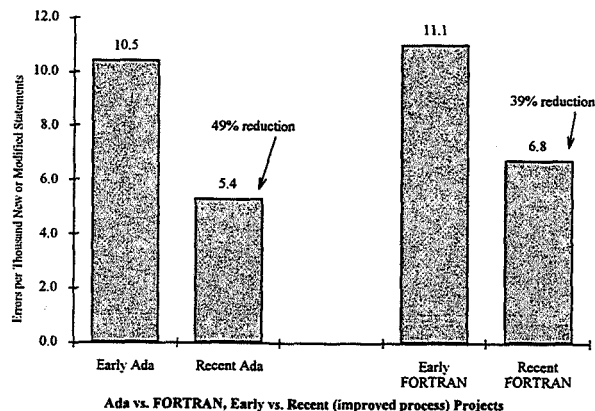


Figure 10. Error Densities on Early and Recent Ada and FORTRAN Projects

The densities shown are based on only the new and modified code (verbatim reused code was not included in the denominator), so these reductions cannot be attributed to reuse. Instead, the reduced error rate is attributed to improvements in the development process that were instituted on all FDD projects during this period. These improvements included the use of object-oriented or encapsulated designs and the use of structured code reading and inspections. The fact that these process improvements were applied to projects in both languages is reflected by the similarity in the error-density reductions observed.

Performance

System performance was not an explicitly stated goal for the programs developed in Ada, but it turned out to be a major issue. By 1985, the programmers in the FDD had achieved such proficiency with FORTRAN software design and implementation that even the most complex flight dynamics systems performed adequately without any special attention being paid to performance issues during design. Thus, performance had become an implicit expectation and was not addressed in software requirements, designs, or test plans.

Figure 11 depicts the relative response times of the delivered simulators between 1984 and 1993, where the response time indicates the wall-clock time required to simulate an interval of data (hours responding per hour of data). A smaller response time indicates better performance. The figure reveals that the first Ada simulator performed very poorly compared with predecessor FORTRAN simulators. Because Ada language benchmarks had shown that Ada executed as fast as equivalent FORTRAN programs and because performance was not an explicit goal, developers of the first Ada project paid little attention to performance. Instead they focused on learning the language and developing reusable, object-oriented software. Predictably, as novice users of this fairly complex language, they did not produce an optimum design or implementation. However, their system was delivered for operational use, so the FDD users' first encounter with an Ada system was negative. This impression was accentuated by the fact that, because of scaled-down processing requirements, the FORTRAN simulator delivered immediately before the first Ada simulator was the fastest simulator ever delivered in this environment.

In retrospect, it appears that attempts to maximize use of OOD while lacking extensive experience with the technology probably contributed more to the initial

poor performance than did Ada. This points to a basic flaw in the approach taken to the evaluation of this new technology: Conflicting goals had been established for Ada by combining its study with the use of OOD. It was then difficult to separate the effects of the language from the effects of OOD techniques, resulting in the language being faulted for the run-time overhead caused by data access procedures and by multiple layers of abstraction.

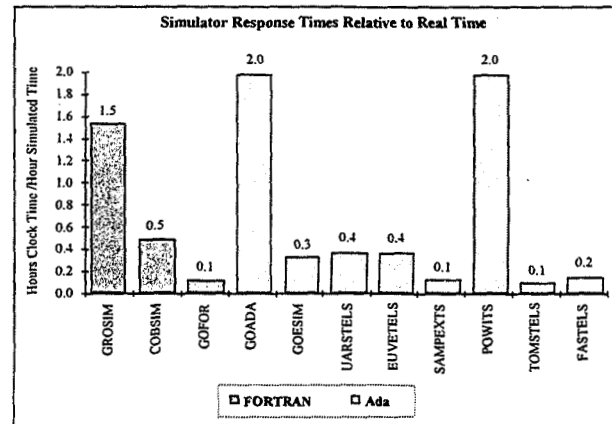


Figure 11. Performance Times of Ada and FORTRAN Simulators

In addition to the overhead from OOD, the 1990 Ada performance study [8] revealed that some of the coding techniques practiced in FORTRAN to achieve high efficiency actually worked against efficiency in Ada, and that some of the data structures around which the designs were built were handled very inefficiently by the DEC Ada compiler. The study resulted in a set of Ada efficiency guidelines [9] for both design and code, which are now being followed for all new Ada systems. Interestingly, to comply with those guidelines, the last two Ada projects in Figure 11 had to forgo a certain amount of reuse (compare with Figure 3). (The slower POWITS simulator was completed before these guidelines were available, and it also had considerably more complex processing requirements as well as other complications.)

As shown in Figure 11, a typical Ada simulator now performs better than most of the earlier FORTRAN simulators. First impressions are very important, though, and some FDD programmers and users still hold the perception that Ada has performance problems and that systems demanding high performance should not be implemented in Ada. Recent impressions have been more favorable toward Ada, however. In fact, subjective data collected about Ada simulator performance reveal that those programmers and users

with recent Ada experience have no complaints about performance. Another indication that performance is no longer an issue is that performance benchmarks are no longer run for the Ada products, although current performance requirements are more demanding.

Qualitative Analysis

Despite the promising quantitative results that accrued from the use of Ada, the adoption of Ada at the FDD was slower, more difficult, and less widespread than expected. As is often the case with technology infusion, several external and internal subjective factors impeded the FDD's transition to Ada. Factors such as the limited availability of Ada compilers and tools, negative feedback from the users of the developed systems, and an adverse and vehement minority opinion within the software development organization all had detrimental effects on the adoption of Ada. This section discusses these factors and their impacts on the goal of transitioning to Ada.

Vendor Tools and Support

Finding adequate vendor tools to support Ada development in the FDD was a major obstacle. In 1985, when the FDD began its work with Ada, most computer vendors were either actively developing Ada compilers and development environments or had announced plans to do so. The FDD believed that, within a few years, vendor tools would be widely available for Ada. However, consistently usable Ada development environments and reliable Ada compilers never became available across the platforms used at the FDD to develop and execute software systems.

With only one small exception, all the Ada projects at the FDD were developed using DEC Ada on VAX minicomputers. The DEC/Ada products available on the VAX platforms were rated by FDD developers as being sufficient to enable viable Ada development. However, 80% of the software developed in the FDD must execute on the standard operational environment, which was an IBM mainframe during this period. Traditionally, FDD systems have been developed on their target platforms because this simplifies testing and deployment. Unfortunately, an adequate Ada development environment was never found for the IBM mainframe.

In search of a solution to mainframe development, the FDD conducted three studies between 1989 and 1992, all of which declared the IBM mainframe environments unfit for Ada software development or deployment. In 1989, the FDD evaluated three compilers [10] and

selected one for purchase and further study. Somewhat discouraged by this study, which rated the best compiler as having only marginal performance for flight dynamics computations and no development tools, the FDD investigated an alternative approach.

Because Ada was touted to be highly portable, the FDD conducted an alternative portability study to determine whether systems could be developed in Ada on the VAX and then transported to the mainframe for operational use. This study [11] ported one of the existing operational Ada simulators from the VAX to the IBM mainframe using the Alsys IBM Ada compiler, version 3.6. The study found that relatively few software changes were required and that the resulting system performed adequately on the mainframe, but that rehosting was extremely difficult because of compiler problems and the lack of diagnostic tools and library management tools. Although rehosting the system required only a small amount of effort, it took nearly as much calendar time as was needed to develop the system from scratch, due to the problems encountered.

An alternate approach would have been for the FDD to purchase a cross-platform development environment, such as the Rational Ada. This cross-targeting strategy, developed to solve many of the problems associated with delivering Ada software on mainframes and other platforms with inadequate Ada environments, would have involved purchasing additional hardware as well as software tools and licenses. Given that the VAX provided a viable Ada development environment, the FDD did not seriously consider converting to the cross platform development environment. Additionally, the cost to use a Rational environment in 1991 would have come to about \$35K in hardware and software per seat, an amount which the FDD deemed excessive.

In the fall of 1992, the FDD again conducted a compiler evaluation on what were supposed to be greatly improved products. This study [12] used the ported simulator as one of its benchmarks and ended up selecting a different compiler than the earlier study. Although the chosen compiler performed better than other candidates and was accompanied by a modest tool set, the study warned against using it to develop real-time or large-scale FDD systems because of its inefficient compiling and binding performance, immature error handling, and poor performance of file input/output. Only in late 1993 did the FDD achieve some limited success developing a small utility in Ada on an IBM RS-6000 workstation and then porting and deploying the software on the mainframe. This

approach to developing Ada systems for mainframe operation is the first to show any real promise. In addition, this has been the only instance of Ada software development on a workstation platform. While the FDD had hoped to begin earlier investigating the appropriateness of using workstations to develop and execute Ada systems, the cost of suitable software development environments on workstation platforms has been prohibitive.

In addition to its disappointment with the mainframe development environments, the FDD also experienced only qualified success using Ada to develop an embedded system. The FDD's R&D effort to develop an embedded application on a Texas Instruments 1750A machine using the TARTAN Ada compiler led to interface problems between the hardware and software. Ultimately, the lack of diagnostic tools contributed to insoluble problems that resulted in an end product with reduced capability. This experience added to the general feeling among FDD developers that the level of vendor support available was unsatisfactory for viable Ada development.

Ada Perspectives Within the FDD

Technology transfer of any software engineering technology involves people: users, software developers, and managers. The introduction and usage of Ada within the FDD sparked much controversy. The independent assessment team sought to determine the impact of the Ada technology on the people of the FDD and to understand the degree to which the attitudes of the various groups impeded or facilitated the infusion of Ada in the FDD. This section presents the key findings from interviews and surveys conducted during the independent assessment.

Users' Perspective

As mentioned earlier, the SEL conducted a performance study in 1990 largely because feedback from the user community indicated that the Ada systems were not as fast as their FORTRAN predecessors and were therefore unacceptable. A survey of 18 users taken in late 1991 showed that performance ideals varied considerably among the users of the satellite simulators. Performance goals ranging from one-quarter real time to 15 times real time were cited by the users, with the most frequently cited performance goal being at least real time (where the simulation of 1 hour of data takes 1 hour of wall-clock time). Most of the *recent* Ada simulators have exceeded this goal; however, when the 1991 survey

was taken, only SAMPEXTS, with a simulation speed of about 3:1, clearly exceeded the 1:1 benchmark. Although the users realized that the complexity of a simulation and the speed and availability of the hardware also affected performance, most blamed the simulators' poor performance on the Ada language itself.

More recent feedback from the users of the Ada telemetry simulators has been entirely favorable. Performance results ranging from 5 times real time to as much as 15 times real time are now being reported. In fact, performance is no longer even considered an issue to the users of the Ada simulators. This is in stark contrast with the situation reported even as recently as 1992. In fact, a major motivation for conducting the independent assessment was to determine the future course of action to address the problems encountered using Ada at the FDD. Apparently, the efforts of the software developers to focus on and improve performance of the TOMSTELS and FASTELS simulators in particular was well worth the sacrifice in reuse.

Developers' Perspective

As part of the study, the independent assessment team conducted two surveys to gather insight into the perspective of the software development staff. The first survey addressed those developers who had direct exposure to Ada, those who either used Ada on the job or attended Ada training, to measure their attitude about the language. The second survey, which addressed the total FDD software development population, measured how each respondent felt about the future of Ada in the FDD. Both provided insight into the overall impact that the introduction of the Ada technology has had on the people in this organization.

The first survey, conducted in mid-1993, gathered information from 35 FDD developers who had been trained in or had developed systems in Ada. Developers were asked which language they would choose for the next simulator project, which language they would choose for the next ground support system, and why. Figure 12 shows the sum of their responses.

Most agreed that Ada should be used for the next simulator but that FORTRAN should be used for the next ground support system, citing the availability of reusable components and architectures as the deciding factors. However, significant minorities in each case recommended use of the language not customarily used for each type of application. The 23% who preferred to use FORTRAN instead of Ada for simulators cited the

complexity of the Ada language and poor performance as reasons to abandon Ada, while the 30% who preferred to use Ada instead of FORTRAN to build the next ground support system felt that Ada was a better language for building larger systems. Interestingly, several of the developers did not care one way or the other about which language they used for software development.

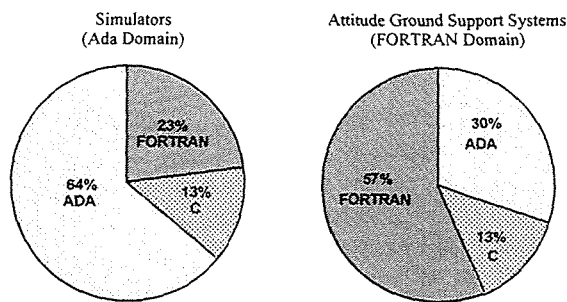


Figure 12. Language Preference for FDD Systems

Nearly all the developers exposed to Ada pointed out that adequate tools are essential for efficient and accurate Ada development, whereas FORTRAN development can be accomplished with little or no external tool support. In particular, they cited the need to have tools to help them with the Ada compilation dependencies that allow Ada's sophisticated error checking during compilation. Interestingly, those who had training followed by on-the-job experience responded positively about Ada, whereas those who had training and no hands-on work experience using Ada had a consistently negative opinion of the language. This indicates that the language is hard (complex) to learn but that, with day-to-day experience, one becomes proficient quickly and experiences the benefits of the language.

As of March 1994, only 25–30% of the development community had been directly exposed to Ada. However, it was clear from interviews and discussions with FDD personnel that there had been a broader impact on the organization as a whole. Two significant minority groups had emerged who were strongly opinionated about language use, one in favor of Ada and the other opposed. Both groups had been fairly vocal and forthcoming with their views throughout the transition to Ada. The second survey was designed to capture the views of the developer community as a whole and to look for a possible effect that these vocal minorities may have had on the remaining group of developers who had not yet been trained in or exposed to Ada.

The second, broader survey, conducted in the spring of 1994, collected responses to questions about basic background information as well as opinions about the use of Ada at the FDD. Background information included job category, FDD experience, computer language experience, and Ada exposure. Ada opinion questions included whether the use of Ada was appropriate at the FDD, whether Ada should be restricted, whether its use should be increased, and whether its use should be decreased. The survey team collected 103 responses from developers (including maintainers and testers), 15 responses from managers, and 7 responses from SEL researchers and others. In order to ensure candid responses, respondents were given the option to return the surveys anonymously.

Approximately half of the developers answered "don't know" or "don't care" to all four Ada opinion questions. Of the half who expressed opinions, a clear majority was positive about the appropriateness of Ada at the FDD. Most felt that the level of usage should remain roughly constant, neither expanding nor reducing the amount or type of application software developed in Ada. Table 2 presents the responses for those who expressed an opinion. The balance of this section summarizes the views of the developers surveyed.

Table 2. Ada Survey Responses for Developers Expressing Opinions

Ada Opinion Questions	All Developers		Developers Without Ada Experience	
	Yes	No	Yes	No
Is Ada appropriate in the FDD?	79%	21%	50%	50%
Should Ada use be restricted?	44%	56%	67%	33%
Should Ada use be increased?	35%	64%	0%	100%
Should Ada use be decreased?	37%	63%	100%	0%

The backgrounds of those with the strongest negative opinions about Ada provided some insight into probable causality. Source of Ada information and knowledge appears to be a key contributor. Of those with the most negative responses, only 1 in 8 had on-the-job experience with Ada. The others had only taken an Ada class or self-studied it, or had no real exposure to Ada at all. (Among those expressing opinions in general, fully one-third had Ada work

experience, confirming that work experience improves one's opinion of the language.) Responses from those developers who received their information about Ada only from others at the FDD indicated that negative opinions about Ada were more likely than positive ones to influence those with no formal Ada exposure.

The survey also revealed a slight negative effect from in-house Ada language training when it was not followed by Ada work experience. This supports anecdotal evidence that the in-house training given at the FDD was detrimental to the typical developer's opinion of the language, while further confirming the positive effects of Ada work experience. Subsequent investigation revealed that responses varied depending on the specific class and instructor who conducted the Ada training. The classes conducted by a trainer from an outside organization were generally better received than those conducted by in-house Ada experts.

The length of time spent at the FDD, the number of years of FORTRAN experience, and the number of computer languages known had no effect on a developer's opinion of Ada. However, a higher than average number of recommendations to decrease the use of Ada came from the customer organization compared with the FDD contractor organization. To obtain a more complete picture of the range and distribution of Ada opinion, including the distribution by organization, the responses to the four questions were converted to composite scores. Positive values were assigned to positive Ada opinions and negative values to negative opinions. Zero values were assigned to "don't know" or "don't care" responses. Figure 13 shows a frequency distribution of the composite scores.

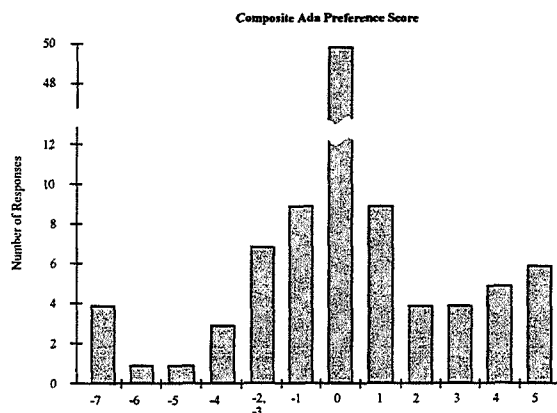


Figure 13. Distribution of Developers' Ada Preference Scores

The tallest bar (in Figure 13), at the neutral score of zero, has been truncated to clarify the shape elsewhere

in the histogram. The tendency for frequencies to diminish outward from the center is interrupted by "bumps" in both tails. These bumps in the curve at both the extreme positive and the extreme negative scores reflect the strongly opinionated and polarized minorities on both sides of the Ada issue. Opinions expressed by the bulk of the respondents, though, fell squarely in the middle, indicating a vast majority having no bias whatsoever.

The contractor organization expressed a more positive overall opinion of Ada than the customer organization. Contractors believed that exposure to and experience with new technologies would make them more marketable and would lead to better future career opportunities. The marketability of Ada developers was confirmed in 1989 when the contractor organization lost several of its most experienced Ada developers after the initial Ada projects were completed. For various reasons, often purely economic ones, several developers chose career moves away from the FDD at a critical time in the Ada transition. Although some of the most knowledgeable Ada developers remained in the FDD, this migration removed a core of Ada experience and opened the door for many new developers to gain Ada experience. Had this exodus not occurred, the subsequent Ada projects would probably have proceeded more smoothly, resulting in a more positive attitude toward Ada among all FDD developers. Nevertheless, the remaining developers in the contractor organization learned first-hand of the opportunities available to their colleagues with Ada experience. Now, in the mid-1990s, C and C++ seem to have replaced Ada as the languages that developers feel will make them more marketable.

The written comments on the survey forms expressed additional observations, perceptions, and points of view about Ada. The most prevalent theme among these comments was the need for adequate tool and vendor support when committing to Ada. Specific references were made to the incompatibility of Ada and the IBM mainframe architecture as well as to the need for reliable vendor support. The lack of readily available packages for interfacing Ada with software toolboxes and other languages was also cited as detrimental.

Next to inadequate tool support, the most commonly mentioned point about Ada was the difficulty experienced in learning and using the language properly. Five developers said either that Ada was hard to learn or that other languages, such as C, were easier to use. Additional specific disappointments included

difficulties with Ada input and output and the complexity of doing true OOD with Ada.

Not unexpectedly, the written comments tended to parallel the Ada opinion scores obtained from the other questions in the survey. The polarity of opinion present at the FDD can be seen here, because the strongest opinions, both negative and positive, were usually held by those at the extreme ends of the opinion score distribution. Overall, there were three unconditional endorsements and six qualified endorsements of the use of Ada in the FDD. On the other hand, five respondents wrote completely negative comments about Ada and another seven were generally pessimistic or skeptical about Ada.

Managers' Perspective

Fifteen managers provided responses to the second Ada opinion survey. About half of the managers (8 of the 15) had Ada exposure but only one had actual on-the-job experience using Ada (one other had managed an Ada project). Classes or seminars in Ada constituted the only exposure to the language among the other six. The half with no Ada exposure (7 of 15) obtained their information from others both within and external to the FDD; only one cited additional sources for his knowledge of Ada, including literature and conferences.

The average management composite score was slightly more positive toward Ada than the average developer score. In general, manager opinions reflected those of their staff. In fact, the division between the customer and contractor organizations was the only clear correlate to the Ada opinion score, with the 5 managers from the customer organization averaging to a net negative opinion and the 10 managers from the contractor organization averaging to a net positive opinion about Ada.

Interestingly, the biggest difference between developer and management opinion came from the substantially greater percentage of managers who favored restricting the use of Ada. This appears to be a sign of caution among managers. When compared with developers, managers did not express any greater or lesser interest in either expanding or reducing the use of Ada; however, they did more often wish to avoid the unrestricted use of the language.

Net Result

Figure 14 depicts the growth of Ada software being delivered each year during the Ada study period. A

sharp decline in the amount of development occurred in late 1990. It was at this point that the FDD had planned to begin developing parts of the larger ground support systems in Ada on the mainframes. However, the results of the early Ada compiler evaluation and the portability studies made it clear that developing on the mainframes, or even developing elsewhere and porting to them, was not feasible. Thus, the growth of new Ada development stalled at this point.

At this same point in time, the FDD's simulation requirements changed, reducing the number of simulators needed to support each spacecraft mission from two to just one. This change resulted in a further reduction in the amount of software slated for development in Ada. The net result was a substantial reduction, instead of the envisioned increase, in the rate of Ada software delivery. The drop in Ada development is even more dramatic when the amount of reused software is eliminated from the totals and only the investment in new Ada code is considered. The flatter, dashed line below the curve for cumulative delivered size in Figure 14 removes the effects of reuse by showing only the number of new and modified lines that were delivered.

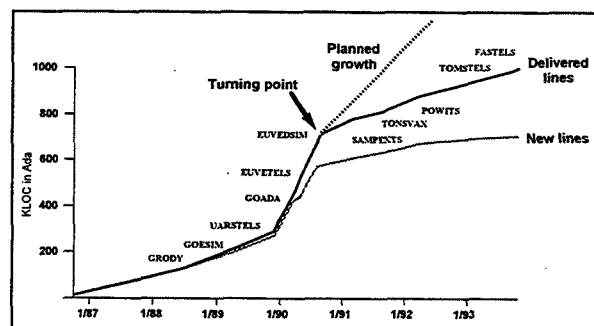


Figure 14. Growth of FDD Ada Software

The unavailability of an adequate Ada development environment on the IBM mainframe was clearly a significant stumbling block for the FDD in its transition to Ada. Had the FDD been able to expand Ada development into the mainframe environment as originally planned, much of the operational software that now exists in FORTRAN would have been written in Ada. Much more of the staff would have gained hands-on work experience in Ada, which, based on the data presented earlier, would have led to a more positive reaction to the language.

If the FDD had continued to use mainframes as its principal operational environment, there would have been no straightforward way to fully transition to Ada. However, the FDD has committed to and has begun

transitioning to open systems for operational support. In the future, software will be developed and deployed on workstations in a networked environment. Thus, a full transition to Ada will depend on the viability of using it for workstation development on a larger scale.

A 1994 FDD internal study found that Ada development environments are very expensive compared with development environments for other languages that support object-oriented development. The typical cost for an adequate Ada development environment for a single workstation seat ranges from \$8.5K to \$17K, depending on the quality and completeness of the tool suite. Conversely, a comparable development environment for C or C++ ranges from \$2K to \$3K per workstation seat. Thus, the high cost of workstation development environments now poses the most serious threat to the future use of Ada in the FDD. The costs have not been reexamined since the 1994 study.

Conclusions

Overall, the FDD benefited greatly from its exposure to and work with Ada. Although, nearly 10 years after Ada's introduction, the FDD uses it to develop only 15–20% of its software, many of the concepts and disciplined software engineering practices associated with Ada have been adopted in the development of all new systems, no matter what language is used. By using object-oriented techniques, such as domain analysis, data abstraction, and information hiding, the FDD has increased its reuse of software by 300%. This in turn has led to reduced mission cost and cycle time for FDD products. Thus, the FDD achieved its original goal of reducing cost and cycle time by maximizing reuse via the introduction and use of the Ada language and OOD.

Although the SEL's assessment of this technology has shown it to be beneficial, it is unlikely that the FDD will fully transition to Ada as its language of choice. The perceived high cost of viable Ada software development environments for workstations continues to be a barrier against using Ada to develop the bulk of the FDD's systems. Up until now, there has been no driving reason to change languages. However, the results documented here show good reason to move away from FORTRAN. As it moves to a distributed workstation hardware environment, the FDD has the opportunity to select a new, cost-effective language(s) for its future. Weighing the tradeoffs between short-term costs, such as software development environments for workstations and initial development, against the

long-term costs of software maintenance, Ada should emerge as a good choice.

The key findings and technology transfer lessons learned from this research and analysis are summarized below. Recommendations are made regarding the future use of Ada in the FDD.

Key Findings

- *Use of Ada and OOD in the FDD:*
 - Increased software reuse by 300%
 - Reduced system cost by 40%
 - Shortened cycle time by 25%
 - Reduced error rates by 62%

By 1990, projects using Ada and OOD were experiencing measured improvement. When compared with the SEL baseline that existed when the Ada assessment began (1985), projects using Ada showed improvement across the board in cost, schedule, and quality as a result of achieving unusually high levels of reuse.

- *The experimentation with Ada and OOD served as a catalyst for many of the improvements seen in the FORTRAN systems during the same period.*

In 1985, Ada was arguably more than just another programming language. Through exposure to the concepts of information hiding, modularity, and packaging for reuse, that which was "more than a language" was adopted, to the extent possible, by the FORTRAN developers as well as by the Ada developers. Anecdotal evidence supports the theory that Ada served to catalyze several language-independent advances in the ways in which software is structured and developed across the organization, and that these benefits have been institutionalized by process improvements. This influence continues today as Ada 95 is used to prototype concepts that are also applied to C++ and Java.

- *FORTRAN systems applying object-oriented concepts also showed significant improvement in reuse. As with the Ada projects, higher reuse led to reduced cycle times and lower error rates on the FORTRAN projects. However, they did not experience similar cost savings; the use of Ada resulted in greater cost reductions for systems with roughly comparable levels of reuse.*

The FORTRAN systems also showed reduced cycle times and improved quality attributable to increased

levels of reuse and the associated process changes when compared with the 1985 baseline. However, the cost reduction was not nearly as significant as with the Ada systems. This was largely due to the effort required to maintain the reusable software. Whereas the use of Ada generics allowed project personnel to reuse code through parameterized instantiation rather than repeated modification, the FORTRAN systems required a separate maintenance team to enhance the reusable components (add new capabilities). Although the separate maintenance team could make the modifications as efficiently as possible (due to familiarity with the code) and the cost of reusing the code from the projects' point of view was virtually nothing, the additional cost of supporting a separate maintenance team nearly negated the savings.

- *Use of Ada resulted in smaller systems to perform more functionality, while generalization increased the size of the FORTRAN systems.*

The use of Ada generics to implement a generalized architecture in the UARSTELS simulator resulted in a system that was 17% smaller than its predecessor (GOESIM) and performed 10% more functionality. Conversely, generalized FORTRAN subsystems are 10-40% larger than earlier single-mission versions. Also, over time, the generalized FORTRAN components have grown as they are enhanced to support new missions, while the size of the generalized Ada components has remained fairly constant.

- *Lack of viable Ada development environments on the FDD's primary development platform severely hampered the transition to Ada.*

When the FDD began using Ada, the availability of vendor tools was of little concern. DoD's mandate that all of its systems be developed in Ada was expected to provide a substantial market for Ada compilers and tools. However, in reality, DoD developed far fewer systems in Ada than expected. This decreased the demand, and vendors lost their incentive to supply Ada support tools. When it became apparent that no vendor planned to provide a full Ada development environment for the IBM mainframe, the FDD had limited options. Because the FDD had just installed a new IBM mainframe, it could not change hardware for at least 5 years. It had neither the money nor the clout (size) that a large company or government agency might have had to offer vendors the incentive to build an Ada environment for the IBM mainframe. Another option available at the time, the Rational development environment, which other IBM-mainframe-based

organizations were using, was prohibitively expensive for the FDD.

Thus, in 1990, when the FDD was ready to expand to full use of Ada, it could not. This essentially stalled the FDD's transition to Ada. Although a small group of people continued to develop simulators and reusable components in Ada, much of the workforce continued to be untouched by the technology. This standstill allowed other languages (such as C, C++) to make advances as viable alternatives to Ada, and allowed opponents of the technology within the workforce to raise doubts about Ada among those who had never been directly exposed to the technology.

- *The high cost of Ada development environments on workstations may deter future use of Ada as the FDD transitions to open systems.*

Today, as the FDD prepares to transition from the mainframe environment to open systems and software development on workstations, the organization is faced with a large investment for new hardware and support software. Ada development environments (compilers and the necessary software development tools) are perceived to cost significantly more (3-8 times more per seat) than development environments for other languages that are commonly used with OOD, such as C++. This poses a financial barrier to the FDD's future use of Ada that should be weighed against the potential savings of building and maintaining systems using Ada.

- *The introduction of Ada sparked much controversy within the FDD. At this time, most of the FDD workforce is lukewarm toward using Ada, with two vocal minorities for and against its continued use. However, most personnel support the use of object-oriented techniques.*

A definite negative attitude toward Ada exists among a small percentage of developers and managers in the FDD who have no direct working experience with Ada. In addition, two small, but vocal groups of people have demonstrated a very strong bias for and against Ada, respectively. Both groups appear to have contributed to the negative bias among the general population: the proponents by overselling the technology and the opponents by negative campaigning. Interestingly, there does not appear to be a corresponding bias against OOD. Nearly all of the developers believe that object-oriented techniques are beneficial and look for ways to apply them, no matter what language they are using.

Technology Transfer Lessons Learned

- *Technology insertion takes a long time, especially when several technologies are combined or when the technology affects the full development life cycle and requires a significant amount of retraining.*

It took approximately 5 years for the FDD to transition to regular routine use of Ada for a particular class of systems. It took nearly 2 years longer to understand the process differences well enough to produce a standard process for Ada projects.

- *First impressions are very important; be careful to understand and set realistic expectations regarding the new technology for everyone affected.*

Negative first impressions caused many problems during the FDD's experience with Ada. Because the developers did not anticipate the impact of the new language and design decisions on system performance, they did not focus on performance requirements during the development of the early systems. Unprepared users were very disappointed in the performance of the early systems and blamed the technology rather than the way in which it had been applied. Today, it is hard to find a dissatisfied user, but it took a lot of effort to overcome the initial impression that Ada was "too slow."

- *Project personnel will focus on and meet the goals set for them at the expense of those not explicitly stated. Be careful to consider all aspects of the new technology when setting goals for pilot projects, and clearly state all goals and their relative priority.*

Each one of the experiments and pilot projects met the goals set for them. However, projects often encountered problems in areas where they sacrificed or overlooked something because of their narrow focus on their primary goal. For example, GRODY personnel explored the new features of the language without paying any attention to system performance. And even after GRODY's poor performance was known, the GOADA and EUVEDSIM teams opted to reuse inefficient code because high reuse was their goal. The GOESIM team sacrificed the use of new Ada features and OO concepts to guarantee delivery on schedule and within budget.

- *New technology advocates are essential to initiate and sustain the technology transfer process. However, if they are not sensitive to the needs and*

concerns of the organization and its developers, they will impede rather than facilitate the process.

The FDD had a few respected technology experts who were very knowledgeable about Ada and OOD and who were enthusiastic proponents of the language. Following their lead, the FDD vigorously pursued Ada and OOD and tried many new ideas that moved the technology's application forward in both industry and the FDD. These technology experts or advocates were expected to assist people who were learning and using the technologies for the first time. However, in some cases, the advocates' zeal for Ada and lack of real project experience made them less sensitive to the concerns of the people who needed to use the new language on real projects. Consequently, they provided help with technical problems, but did not acknowledge and constructively discuss others' frustrations with applying Ada. Gradually people became disillusioned with the technology advocates, stopped going to them for help, and in some cases, began to actively campaign against them. This greatly impeded the technology infusion process.

Technology experts are essential to understanding and applying new technology correctly, but not all are well-suited to the advocate role. Advocates should be chosen carefully and the other technology experts kept in the background. Outside consultants should be used for initial training and coaching, and respected senior personnel and project leaders should be relied on to be coaches after they have been trained and have used the technology on a project.

- *Initial language training is best accomplished by outside vendors. Local training should focus on how to apply the language in the local environment.*

Of the two methods used for institutional Ada training in the FDD, the language courses taught by outside vendors (external to the local FDD/contractor organizations) were more successful. The FDD training experiences indicate that new technology training is best when taught by an instructor who is not known within the organization. That way the technology is not loaded with the extra baggage of personality conflicts or issues such as contractors teaching customers with whom they work on a daily basis. Obviously, local application of the technology should be taught by someone within the local organization. Here it is best to use a senior developer or manager who has learned the technology and applied it on a project, rather than a

technology expert who may lack "real-world" experience using the technology.

Current Usage of Ada in the FDD

In the early 1990s, the FDD began developing the concepts and architecture for its Flight Dynamics Distributed System (FDDS). The FDD based this design on open systems concepts and what had been learned from Ada prototypes and real project experience about the importance of architecture, programming language, and library support on the reconfigurability of reusable software components.

In 1993, the FDD began building this new project support environment and a repository for reusable application code, called the Generalized Software Support (GSS) library. The GSS was expected to facilitate the rapid construction of future flight dynamics ground systems and simulators from large-scale reusable components. Ada 83 was the FDD's language of choice for these reusable components, which support a broad range of flight dynamics applications including attitude support, navigation support, and mission planning.

In March 1995, the Ada study team made the following recommendations:

- *The FDD should continue to use Ada whenever possible. Examples include those systems that reuse existing Ada code and any other projects (or portions of projects) that are expected to be long-lived and can be developed and deployed on an Ada-capable platform.*
- *The FDD should build reusable software in a language that supports object-oriented constructs and should consider using specialized teams of experts to configure the reusable components for each mission. This would likely further improve the efficiency of the reuse process.*
- *The FDD should investigate lower-cost, alternative languages to support object-oriented development on workstations. However, trade-off analyses should consider the cost of software development environments, the efficiency and quality of software development, and the ease and cost of long-term maintenance for the languages under consideration.*

Shortly after the final report was published, the FDD made the following two major decisions:

- *All operational flight dynamics software, new and legacy, will operate on workstations starting in September 1996; the mainframe computers will be removed in the fall of 1996. This means that all new software must be developed on and targeted for a distributed workstation environment.*
- *The FDD selected C++ as its language of choice for future object-oriented software development. This decision was largely a business decision based on the cost and availability of adequate development tools on workstations and the need to translate existing code to C/C++ or Ada 95, whichever language was selected. This decision was also influenced somewhat by the negative bias of the staff toward Ada.*

As a result of these decisions, all operational software will become part of the FDDS. In mid-1995, a massive effort was initiated to port or replace all operational legacy software. Due to the urgency of this task, much of the FORTRAN software is being directly ported to workstations, with any replacement components being implemented in C or C++ as necessary.

As a result of the language decision in 1995, the GSS reusable mission planning component is now being developed in C++, and any existing Ada utilities in the GSS library have been converted to C++ for the project use. However, because 75 % of the GSS attitude support system reusable components had already been developed, the FDD decided to leverage its investment and continue development of the attitude support component set in Ada 83. The reusable attitude component set is scheduled to be completed in July, and two satellite mission ground support systems have been configured to date. Preliminary project results from one mission show improvement in the quality of code delivered to the independent testing team, while the effort and cycle time required to deliver a mission ground support system and simulator using the Ada GSS components remain comparable to previous projects. This is encouraging since the development team had a large learning curve to overcome in first-use validation of the architecture and process for configuring applications from the reusable library components [13]. Projects currently underway are showing significant improvements in effort and cycle time. Unfortunately, no data is available from the C++ projects yet for comparison. This recent project data continues to demonstrate that Ada is a wise choice for building reliable reusable software.

References

1. NASA/GSFC Software Engineering Laboratory, SEL-95-001, *Impact of Ada and Object-Oriented Design in the Flight Dynamics Division at Goddard Space Flight Center*, S. Waligora, J. Bailey, M. Stark, March 1995*
2. ___, SEL-93-003, "Impact of Ada in the Flight Dynamics Division: Excitement and Frustration," J. Bailey, S. Waligora, M. Stark, *Proceedings of the Eighteenth Annual Software Engineering Workshop*, pp. 422-438, December 1993
3. ___, SEL-81-305SP1, *Ada Developers' Supplement to the Recommended Approach*, L. Landis, R. Kester, November 1993*
4. ___, SEL-81-305, *Recommended Approach to Software Development*, L. Landis, S. Waligora, F. McGarry, et al., June 1992 *
5. ___, SEL-91-006, "Experiments in Software Engineering Technology," *Proceedings of the Sixteenth Annual Software Engineering Workshop*, F. McGarry and S. Waligora, December 1991
6. ___, SEL-82-1306, *Annotated Bibliography of Software Engineering Laboratory Literature*, D. Kistler, J. Bristow, D. Smith, November 1994
7. Institute for Defense Analysis, IDA Paper P-2899, "Comparing Ada and FORTRAN Lines of Code: Some Experimental Results," T. Frazier, J. Bailey, M. Young, November 1993
8. NASA/GSFC Software Engineering Laboratory, SEL-91-003, *Ada Performance Study Report*, E. Booth and M. Stark, July 1991
9. Goddard Space Flight Center, Flight Dynamics Division, 552-FDD-91/068R0UD0, *Ada Efficiency Guide*, E. Booth, August 1992
10. ___, *Ada Compilers on the IBM Mainframe (NAS8040) Evaluation Report*, L. Jun, January 1989
11. NASA/GSFC Software Engineering Laboratory, SEL-90-003, *A Study of the Portability of an Ada System in the Software Engineering Laboratory*, L. Jun and S. Valett, June 1990
12. Goddard Space Flight Center, Flight Dynamics Division, IBM Ada/370 (Release 2.0) Compiler Evaluation Report, L. Jun, September 1992, and Intermetrics MVS/Ada Version 8.0 Compiler Evaluation Report, L. Jun, October 1992
13. "The Generalized Support Software Domain Engineering Process: An Object-Oriented Implementation and Reuse Success at Goddard Space Flight Center," S. Condon, R. Hendrick, M. Stark, W. Steger, to be presented at the Conference on Object-Oriented Programming Systems, Languages, and Applications in October 1996.

* Available via the SEL Home Page on the World Wide Web at <http://fdd.gsfc.nasa.gov/seltext.html>

STANDARD BIBLIOGRAPHY OF SEL LITERATURE

The technical papers, memorandums, and documents listed in this bibliography are organized into two groups. The first group is composed of documents issued by the Software Engineering Laboratory (SEL) during its research and development activities. The second group includes materials that were published elsewhere but pertain to SEL activities. The *Annotated Bibliography of Software Engineering Laboratory Literature* contains an abstract for each document and is available via the SEL Products Page at <http://fdd.gsfc.nasa.gov/selprods.html>.

SEL-ORIGINATED DOCUMENTS

SEL-76-001, *Proceedings From the First Summer Software Engineering Workshop*, August 1976

SEL-77-002, *Proceedings From the Second Summer Software Engineering Workshop*, September 1977

SEL-78-005, *Proceedings From the Third Summer Software Engineering Workshop*, September 1978

SEL-78-006, *GSFC Software Engineering Research Requirements Analysis Study*, P. A. Scheffer and C. E. Velez, November 1978

SEL-78-007, *Applicability of the Rayleigh Curve to the SEL Environment*, T. E. Mapp, December 1978

SEL-78-302, *FORTTRAN Static Source Code Analyzer Program (SAP) User's Guide (Revision 3)*, W. J. Decker, W. A. Taylor, et al., July 1986

SEL-79-002, *The Software Engineering Laboratory: Relationship Equations*, K. Freburger and V.R. Basili, May 1979

SEL-79-004, *Evaluation of the Caine, Farber, and Gordon Program Design Language (PDL) in the Goddard Space Flight Center (GSFC) Code 580 Software Design Environment*, C.E. Goorevich, A. L. Green, and W. J. Decker, September 1979

SEL-79-005, *Proceedings From the Fourth Summer Software Engineering Workshop*, November 1979

SEL-80-002, *Multi-Level Expression Design Language-Requirement Level (MEDL-R) System Evaluation*, W. J. Decker and C. E. Goorevich, May 1980

SEL-80-005, *A Study of the Musa Reliability Model*, A. M. Miller, November 1980

SEL-80-006, *Proceedings From the Fifth Annual Software Engineering Workshop*, November 1980

SEL-80-007, *An Appraisal of Selected Cost/Resource Estimation Models for Software Systems*, J.F. Cook and F. E. McGarry, December 1980

SEL-80-008, *Tutorial on Models and Metrics for Software Management and Engineering*, V.R.Basili, 1980

SEL-81-011, *Evaluating Software Development by Analysis of Change Data*, D. M. Weiss, November 1981

SEL-81-012, *The Rayleigh Curve as a Model for Effort Distribution Over the Life of Medium Scale Software Systems*, G. O. Picasso, December 1981

SEL-81-013, *Proceedings of the Sixth Annual Software Engineering Workshop*, December 1981

SEL-81-014, *Automated Collection of Software Engineering Data in the Software Engineering Laboratory (SEL)*, A. L. Green, W. J. Decker, and F. E. McGarry, September 1981

SEL-81-101, *Guide to Data Collection*, V. E. Church, D. N. Card, F. E. McGarry, et al., August 1982

SEL-81-110, *Evaluation of an Independent Verification and Validation (IV&V) Methodology for Flight Dynamics*, G. Page, F. E. McGarry, and D. N. Card, June 1985

SEL-81-305, *Recommended Approach to Software Development*, L. Landis, S. Waligora, F.E.1McGarry, et al., June 1992

SEL-81-305SP1, *Ada Developers' Supplement to the Recommended Approach*, R. Kester and L.Landis, November 1993

SEL-82-001, *Evaluation of Management Measures of Software Development*, G. Page, D.N.Card, and F. E. McGarry, September 1982, vols. 1 and 2

SEL-82-004, *Collected Software Engineering Papers: Volume 1*, July 1982

SEL-82-007, *Proceedings of the Seventh Annual Software Engineering Workshop*, December 1982

SEL-82-008, *Evaluating Software Development by Analysis of Changes: The Data From the Software Engineering Laboratory*, V. R. Basili and D. M. Weiss, December 1982

SEL-82-102, *FORTTRAN Static Source Code Analyzer Program (SAP) System Description (Revision 1)*, W. A. Taylor and W. J. Decker, April 1985

SEL-82-105, *Glossary of Software Engineering Laboratory Terms*, T. A. Babst, M. G. Rohleder, and F. E. McGarry, October 1983

SEL-82-1306, *Annotated Bibliography of Software Engineering Laboratory Literature*, D. Kistler, J. Bristow, and D. Smith, November 1994

SEL-83-001, *An Approach to Software Cost Estimation*, F. E. McGarry, G. Page, D. N. Card, et al., February 1984

SEL-83-002, *Measures and Metrics for Software Development*, D. N. Card, F. E. McGarry, G.Page, et al., March 1984

SEL-83-003, *Collected Software Engineering Papers: Volume II*, November 1983

SEL-83-007, *Proceedings of the Eighth Annual Software Engineering Workshop*, November 1983

SEL-83-106, *Monitoring Software Development Through Dynamic Variables (Revision 1)*, C.W.Doerflinger, November 1989

SEL-84-003, *Investigation of Specification Measures for the Software Engineering Laboratory (SEL)*, W. W. Agresti, V. E. Church, and F. E. McGarry, December 1984

SEL-84-004, *Proceedings of the Ninth Annual Software Engineering Workshop*, November 1984

SEL-84-101, *Manager's Handbook for Software Development (Revision 1)*, L. Landis, F.E.McGarry, S. Waligora, et al., November 1990

SEL-85-001, *A Comparison of Software Verification Techniques*, D. N. Card, R. W. Selby, Jr., F.E. McGarry, et al., April 1985

SEL-85-002, *Ada Training Evaluation and Recommendations From the Gamma Ray Observatory Ada Development Team*, R. Murphy and M. Stark, October 1985

SEL-85-003, *Collected Software Engineering Papers: Volume III*, November 1985

SEL-85-004, *Evaluations of Software Technologies: Testing, CLEANROOM, and Metrics*, R.W.Selby, Jr., and V. R. Basili, May 1985

SEL-85-005, *Software Verification and Testing*, D. N. Card, E. Edwards, F. McGarry, and C.Antle, December 1985

SEL-85-006, *Proceedings of the Tenth Annual Software Engineering Workshop*, December 1985

SEL-86-001, *Programmer's Handbook for Flight Dynamics Software Development*, R. Wood and E. Edwards, March 1986

SEL-86-002, *General Object-Oriented Software Development*, E. Seidewitz and M. Stark, August 1986

SEL-86-003, *Flight Dynamics System Software Development Environment (FDS/SDE) Tutorial*, J. Buell and P. Myers, July 1986

SEL-86-004, *Collected Software Engineering Papers: Volume IV*, November 1986

SEL-86-005, *Measuring Software Design*, D. N. Card et al., November 1986

SEL-86-006, *Proceedings of the Eleventh Annual Software Engineering Workshop*, December 1986

SEL-87-001, *Product Assurance Policies and Procedures for Flight Dynamics Software Development*, S. Perry et al., March 1987

SEL-87-002, *Ada[®] Style Guide (Version 1.1)*, E. Seidewitz et al., May 1987

SEL-87-003, *Guidelines for Applying the Composite Specification Model (CSM)*, W. W. Agresti, June 1987

SEL-87-004, *Assessing the Ada® Design Process and Its Implications: A Case Study*, S. Godfrey, C. Brophy, et al., July 1987

SEL-87-009, *Collected Software Engineering Papers: Volume V*, November 1987

SEL-87-010, *Proceedings of the Twelfth Annual Software Engineering Workshop*, December 1987

SEL-88-001, *System Testing of a Production Ada Project: The GRODY Study*, J. Seigle, L. Esker, and Y. Shi, November 1988

SEL-88-002, *Collected Software Engineering Papers: Volume VI*, November 1988

SEL-88-003, *Evolution of Ada Technology in the Flight Dynamics Area: Design Phase Analysis*, K. Quimby and L. Esker, December 1988

SEL-88-004, *Proceedings of the Thirteenth Annual Software Engineering Workshop*, November 1988

SEL-88-005, *Proceedings of the First NASA Ada User's Symposium*, December 1988

SEL-89-002, *Implementation of a Production Ada Project: The GRODY Study*, S. Godfrey and C. Brophy, September 1989

SEL-89-004, *Evolution of Ada Technology in the Flight Dynamics Area: Implementation/Testing Phase Analysis*, K. Quimby, L. Esker, L. Smith, M. Stark, and F. McGarry, November 1989

SEL-89-005, *Lessons Learned in the Transition to Ada From FORTRAN at NASA/Goddard*, C. Brophy, November 1989

SEL-89-006, *Collected Software Engineering Papers: Volume VII*, November 1989

SEL-89-007, *Proceedings of the Fourteenth Annual Software Engineering Workshop*, November 1989

SEL-89-008, *Proceedings of the Second NASA Ada Users' Symposium*, November 1989

SEL-89-103, *Software Management Environment (SME) Concepts and Architecture (Revision 1)*, R. Hendrick, D. Kistler, and J. Valett, September 1992

SEL-89-301, *Software Engineering Laboratory (SEL) Database Organization and User's Guide (Revision 3)*, L. Morusiewicz, February 1995

SEL-90-001, *Database Access Manager for the Software Engineering Laboratory (DAMSEL) User's Guide*, M. Buhler, K. Pumphrey, and D. Spiegel, March 1990

SEL-90-002, *The Cleanroom Case Study in the Software Engineering Laboratory: Project Description and Early Analysis*, S. Green et al., March 1990

SEL-90-003, *A Study of the Portability of an Ada System in the Software Engineering Laboratory (SEL)*, L. O. Jun and S. R. Valett, June 1990

SEL-90-004, *Gamma Ray Observatory Dynamics Simulator in Ada (GRODY) Experiment Summary*, T. McDermott and M. Stark, September 1990

SEL-90-005, *Collected Software Engineering Papers: Volume VIII*, November 1990

SEL-90-006, *Proceedings of the Fifteenth Annual Software Engineering Workshop*, November 1990

SEL-91-001, *Software Engineering Laboratory (SEL) Relationships, Models, and Management Rules*, W. Decker, R. Hendrick, and J. Valett, February 1991

SEL-91-003, *Software Engineering Laboratory (SEL) Ada Performance Study Report*, E.W.Booth and M. E. Stark, July 1991

SEL-91-004, *Software Engineering Laboratory (SEL) Cleanroom Process Model*, S. Green, November 1991

SEL-91-005, *Collected Software Engineering Papers: Volume IX*, November 1991

SEL-91-006, *Proceedings of the Sixteenth Annual Software Engineering Workshop*, December 1991

SEL-91-102, *Software Engineering Laboratory (SEL) Data and Information Policy (Revision 1)*, F. McGarry, August 1991

SEL-92-001, *Software Management Environment (SME) Installation Guide*, D. Kistler and K.Jeletic, January 1992

SEL-92-002, *Data Collection Procedures for the Software Engineering Laboratory (SEL) Database*, G. Heller, J. Valett, and M. Wild, March 1992

SEL-92-003, *Collected Software Engineering Papers: Volume X*, November 1992

SEL-92-004, *Proceedings of the Seventeenth Annual Software Engineering Workshop*, December 1992

SEL-93-001, *Collected Software Engineering Papers: Volume XI*, November 1993

SEL-93-002, *Cost and Schedule Estimation Study Report*, S. Condon, M. Regardie, M. Stark, et al., November 1993

SEL-93-003, *Proceedings of the Eighteenth Annual Software Engineering Workshop*, December 1993

SEL-94-001, *Software Management Environment (SME) Components and Algorithms*, R.Hendrick, D. Kistler, and J. Valett, February 1994

SEL-94-003, *C Style Guide*, J. Doland and J. Valett, August 1994

SEL-94-004, *Collected Software Engineering Papers: Volume XII*, November 1994

SEL-94-005, *An Overview of the Software Engineering Laboratory*, F. McGarry, G. Page, V.Basili, et al., December 1994

SEL-94-006, *Proceedings of the Nineteenth Annual Software Engineering Workshop*, December 1994

SEL-94-102, *Software Measurement Guidebook (Revision 1)*, M. Bassman, F. McGarry, R. Pajerski, June 1995

SEL-95-001, *Impact of Ada in the Flight Dynamics Division at Goddard Space Flight Center*, S. Waligora, J. Bailey, M. Stark, March 1995

SEL-95-003, *Collected Software Engineering Papers: Volume XIII*, November 1995

SEL-96-001, *Collected Software Engineering Papers: Volume XIV*, October 1996

SEL-RELATED LITERATURE

¹⁰Abd-El-Hafiz, S. K., V. R. Basili, and G. Caldiera, "Towards Automated Support for Extraction of Reusable Components," *Proceedings of the IEEE Conference on Software Maintenance-1991 (CSM 91)*, October 1991

⁴Agresti, W. W., V. E. Church, D. N. Card, and P. L. Lo, "Designing With Ada for Satellite Simulation: A Case Study," *Proceedings of the First International Symposium on Ada for the NASA Space Station*, June 1986

²Agresti, W. W., F. E. McGarry, D. N. Card, et al., "Measuring Software Technology," *Program Transformation and Programming Environments*. New York: Springer-Verlag, 1984

¹Bailey, J. W., and V. R. Basili, "A Meta-Model for Software Development Resource Expenditures," *Proceedings of the Fifth International Conference on Software Engineering*. New York: IEEE Computer Society Press, 1981

⁸Bailey, J. W., and V. R. Basili, "Software Reclamation: Improving Post-Development Reusability," *Proceedings of the Eighth Annual National Conference on Ada Technology*, March 1990

¹⁰Bailey, J. W., and V. R. Basili, "The Software-Cycle Model for Re-Engineering and Reuse," *Proceedings of the ACM Tri-Ada 91 Conference*, October 1991

¹Basili, V. R., "Models and Metrics for Software Management and Engineering," *ASME Advances in Computer Technology*, January 1980, vol. 1

Basili, V. R., *Tutorial on Models and Metrics for Software Management and Engineering*. New York: IEEE Computer Society Press, 1980 (also designated SEL-80-008)

³Basili, V. R., "Quantitative Evaluation of Software Methodology," *Proceedings of the First Pan-Pacific Computer Conference*, September 1985

⁷Basili, V. R., *Maintenance = Reuse-Oriented Software Development*, University of Maryland, Technical Report TR-2244, May 1989

⁷Basili, V. R., *Software Development: A Paradigm for the Future*, University of Maryland, Technical Report TR-2263, June 1989

⁸Basili, V. R., "Viewing Maintenance of Reuse-Oriented Software Development," *IEEE Software*, January 1990

¹³Basili, V. R., "The Experience Factory and Its Relationship to Other Quality Approaches," *Advances in Computers*, vol. 41, Academic Press, Incorporated, 1995

¹Basili, V. R., and J. Beane, "Can the Parr Curve Help With Manpower Distribution and Resource Estimation Problems?," *Journal of Systems and Software*, February 1981, vol. 2, no. 1

¹³Basili, V. R., L. Briand, and W. L. Melo, *A Validation of Object-Oriented Design Metrics*, University of Maryland, Computer Science Technical Report, CS-TR-3443, UMIACS-TR-95-40, April 1995

¹³Basili, V. R., and G. Caldiera, *The Experience Factory Strategy and Practice*, University of Maryland, Computer Science Technical Report, CS-TR-3483, UMIACS-TR-95-67, May 1995

⁹Basili, V. R., G. Caldiera, and G. Cantone, "A Reference Architecture for the Component Factory," *ACM Transactions on Software Engineering and Methodology*, January 1992

¹⁰Basili, V., G. Caldiera, F. McGarry, et al., "The Software Engineering Laboratory—An Operational Software Experience Factory," *Proceedings of the Fourteenth International Conference on Software Engineering (ICSE 92)*, May 1992

¹Basili, V. R., and K. Freburger, "Programming Measurement and Estimation in the Software Engineering Laboratory," *Journal of Systems and Software*, February 1981, vol. 2, no. 1

¹²Basili, V. R., and S. Green, "Software Process Evolution at the SEL," *IEEE Software*, July 1994, pp.58–66

³Basili, V. R., and N. M. Panlilio-Yap, "Finding Relationships Between Effort and Other Variables in the SEL," *Proceedings of the International Computer Software and Applications Conference*, October 1985

⁴Basili, V. R., and D. Patnaik, *A Study on Fault Prediction and Reliability Assessment in the SEL Environment*, University of Maryland, Technical Report TR-1699, August 1986

²Basili, V. R., and B. T. Perricone, "Software Errors and Complexity: An Empirical Investigation," *Communications of the ACM*, January 1984, vol. 27, no. 1

¹Basili, V. R., and T. Phillips, "Evaluating and Comparing Software Metrics in the Software Engineering Laboratory," *Proceedings of the ACM SIGMETRICS Symposium/Workshop: Quality Metrics*, March 1981

³Basili, V. R., and C. L. Ramsey, "ARROWSMITH-P—A Prototype Expert System for Software Engineering Management," *Proceedings of the IEEE/MITRE Expert Systems in Government Symposium*, October 1985

Basili, V. R., and J. Ramsey, *Structural Coverage of Functional Testing*, University of Maryland, Technical Report TR-1442, September 1984

Basili, V. R., and R. Reiter, "Evaluating Automatable Measures for Software Development," *Proceedings of the Workshop on Quantitative Software Models for Reliability, Complexity, and Cost*. New York: IEEE Computer Society Press, 1979

⁵Basili, V. R., and H. D. Rombach, "Tailoring the Software Process to Project Goals and Environments," *Proceedings of the 9th International Conference on Software Engineering*, March 1987

⁵Basili, V. R., and H. D. Rombach, "TAME: Tailoring an Ada Measurement Environment," *Proceedings of the Joint Ada Conference*, March 1987

⁵Basili, V. R., and H. D. Rombach, "TAME: Integrating Measurement Into Software Environments," University of Maryland, Technical Report TR-1764, June 1987

⁶Basili, V. R., and H. D. Rombach, "The TAME Project: Towards Improvement-Oriented Software Environments," *IEEE Transactions on Software Engineering*, June 1988

⁷Basili, V. R., and H. D. Rombach, *Towards A Comprehensive Framework for Reuse: A Reuse-Enabling Software Evolution Environment*, University of Maryland, Technical Report TR-2158, December 1988

⁸Basili, V. R., and H. D. Rombach, *Towards A Comprehensive Framework for Reuse: Model-Based Reuse Characterization Schemes*, University of Maryland, Technical Report TR-2446, April 1990

⁹Basili, V. R., and H. D. Rombach, "Support for Comprehensive Reuse," *Software Engineering Journal*, September 1991

³Basili, V. R., and R. W. Selby, Jr., "Calculation and Use of an Environment's Characteristic Software Metric Set," *Proceedings of the Eighth International Conference on Software Engineering*. New York: IEEE Computer Society Press, 1985

Basili, V. R., and R. W. Selby, "Comparing the Effectiveness of Software Testing Strategies," *IEEE Transactions on Software Engineering*, December 1987

³Basili, V. R., and R. W. Selby, Jr., "Four Applications of a Software Data Collection and Analysis Methodology," *Proceedings of the NATO Advanced Study Institute*, August 1985

⁵Basili, V. R., and R. Selby, "Comparing the Effectiveness of Software Testing Strategies," *IEEE Transactions on Software Engineering*, December 1987

⁹Basili, V. R., and R. W. Selby, "Paradigms for Experimentation and Empirical Studies in Software Engineering," *Reliability Engineering and System Safety*, January 1991

⁴Basili, V. R., R. W. Selby, Jr., and D. H. Hutchens, "Experimentation in Software Engineering," *IEEE Transactions on Software Engineering*, July 1986

²Basili, V. R., R. W. Selby, and T. Phillips, "Metric Analysis and Data Validation Across FORTRAN Projects," *IEEE Transactions on Software Engineering*, November 1983

²Basili, V. R., and D. M. Weiss, *A Methodology for Collecting Valid Software Engineering Data*, University of Maryland, Technical Report TR-1235, December 1982

³Basili, V. R., and D. M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," *IEEE Transactions on Software Engineering*, November 1984

¹Basili, V. R., and M. V. Zelkowitz, "The Software Engineering Laboratory: Objectives," *Proceedings of the Fifteenth Annual Conference on Computer Personnel Research*, August 1977

Basili, V. R., and M. V. Zelkowitz, "Designing a Software Measurement Experiment," *Proceedings of the Software Life Cycle Management Workshop*, September 1977

¹Basili, V. R., and M. V. Zelkowitz, "Operation of the Software Engineering Laboratory," *Proceedings of the Second Software Life Cycle Management Workshop*, August 1978

¹Basili, V. R., and M. V. Zelkowitz, "Measuring Software Development Characteristics in the Local Environment," *Computers and Structures*, August 1978, vol. 10

Basili, V. R., and M. V. Zelkowitz, "Analyzing Medium Scale Software Development," *Proceedings of the Third International Conference on Software Engineering*. New York: IEEE Computer Society Press, 1978

¹³Basili, V., M. Zelkowitz, F. McGarry, G. Page, S. Waligora, and R. Pajerski, "SEL's Software Process-Improvement Program," *IEEE Software*, vol. 12, no. 6, November 1995, pp. 83-87

¹⁴Basili, V., C. Seaman, "Communication and Organization in Software Development: An Empirical Study"

¹⁴Basili, V., S. Green, O. Laitenberger, F. Shull, S. Sorumgard, and M. Zelkowitz, "The Empirical Investigation of Perspective-Based Reading"

¹⁴Basili, V., "Evolving and Packaging Reading Technologies," *Proceedings of the Third International Conference on Achieving Quality in Software, Florence, Italy, January 1996*

¹⁴Basili, V., "The Role of Experimentation in Software Engineering: Past, Current, and Future," *Proceedings of the Eighteenth Annual Conference on Software Engineering (ICSE-18)*, March 1996

¹⁴Basili, V., L. Briand, S. Condon, Y. Kim, W. Melo and J. Valett, "Understanding and Predicting the Process of Software Maintenance Releases," *Proceedings of the Eighteenth Annual Conference on Software Engineering (ICSE-18)*, March 1996

Bassman, M. J., F. McGarry, and R. Pajerski, *Software Measurement Guidebook*, NASA-GB-001-94, Software Engineering Program, July 1994

⁹Booth, E. W., and M. E. Stark, "Designing Configurable Software: COMPASS Implementation Concepts," *Proceedings of Tri-Ada 1991*, October 1991

¹⁰Booth, E. W., and M. E. Stark, "Software Engineering Laboratory Ada Performance Study—Results and Implications," *Proceedings of the Fourth Annual NASA Ada User's Symposium*, April 1992

- ¹⁰Briand, L. C., and V. R. Basili, "A Classification Procedure for the Effective Management of Changes During the Maintenance Process," *Proceedings of the 1992 IEEE Conference on Software Maintenance (CSM 92)*, November 1992
- ¹⁰Briand, L. C., V. R. Basili, and C. J. Hetmanski, "Providing an Empirical Basis for Optimizing the Verification and Testing Phases of Software Development," *Proceedings of the Third IEEE International Symposium on Software Reliability Engineering (ISSRE 92)*, October 1992
- ¹¹Briand, L. C., V. R. Basili, and C. J. Hetmanski, *Developing Interpretable Models with Optimized Set Reduction for Identifying High Risk Software Components*, University of Maryland, Technical Report TR-3048, March 1993
- ¹²Briand, L. C., V. R. Basili, Y. Kim, and D. R. Squier, "A Change Analysis Process to Characterize Software Maintenance Projects," *Proceedings of the International Conference on Software Maintenance*, Victoria, British Columbia, Canada, September 19–23, 1994, pp. 38–49
- ⁹Briand, L. C., V. R. Basili, and W. M. Thomas, *A Pattern Recognition Approach for Software Engineering Data Analysis*, University of Maryland, Technical Report TR-2672, May 1991
- ¹³Briand, L., W. Melo, C. Seaman, and V. Basili, "Characterizing and Assessing a Large-Scale Software Maintenance Organization," *Proceedings of the 17th International Conference on Software Engineering*, Seattle, Washington, U.S.A., April 23–30, 1995
- ¹¹Briand, L. C., S. Morasca, and V. R. Basili, "Measuring and Assessing Maintainability at the End of High Level Design," *Proceedings of the 1993 IEEE Conference on Software Maintenance (CSM 93)*, November 1993
- ¹²Briand, L., S. Morasca, and V. R. Basili, *Defining and Validating High-Level Design Metrics*, University of Maryland, Computer Science Technical Report, CS-TR-3301, UMIACS-TR-94-75, June 1994
- ¹³Briand, L., S. Morasca, and V. R. Basili, *Property-based Software Engineering Measurement*, University of Maryland, Computer Science Technical Report, CS-TR-3368, UMIACS-TR-94-119, November 1994
- ¹³Briand, L., S. Morasca, and V. R. Basili, *Goal-Driven Definition of Product Metrics Based on Properties*, University of Maryland, Computer Science Technical Report, CS-TR-3346, UMIACS-TR-94-106, December 1994
- ¹¹Briand, L. C., W. M. Thomas, and C. J. Hetmanski, "Modeling and Managing Risk Early in Software Development," *Proceedings of the Fifteenth International Conference on Software Engineering (ICSE 93)*, May 1993
- ¹⁴Briand, L., Y. Kim, W. Melo, C. Seaman, V. Basili, "Qualitative Analysis for Maintenance Process Assessment"
- ⁵Brophy, C. E., W. W. Agresti, and V. R. Basili, "Lessons Learned in Use of Ada-Oriented Design Methods," *Proceedings of the Joint Ada Conference*, March 1987

⁶Brophy, C. E., S. Godfrey, W. W. Agresti, and V. R. Basili, "Lessons Learned in the Implementation Phase of a Large Ada Project," *Proceedings of the Washington Ada Technical Conference*, March 1988

²Card, D. N., "Early Estimation of Resource Expenditures and Program Size," Computer Sciences Corporation, Technical Memorandum, June 1982

²Card, D. N., "Comparison of Regression Modeling Techniques for Resource Estimation," Computer Sciences Corporation, Technical Memorandum, November 1982

³Card, D. N., "A Software Technology Evaluation Program," *Annais do XVIII Congresso Nacional de Informatica*, October 1985

⁵Card, D. N., and W. W. Agresti, "Resolving the Software Science Anomaly," *Journal of Systems and Software*, 1987

⁶Card, D. N., and W. W. Agresti, "Measuring Software Design Complexity," *Journal of Systems and Software*, June 1988

⁴Card, D. N., V. E. Church, and W. W. Agresti, "An Empirical Study of Software Design Practices," *IEEE Transactions on Software Engineering*, February 1986

Card, D. N., V. E. Church, W. W. Agresti, and Q. L. Jordan, "A Software Engineering View of Flight Dynamics Analysis System," Parts I and II, Computer Sciences Corporation, Technical Memorandum, February 1984

Card, D. N., Q. L. Jordan, and V. E. Church, "Characteristics of FORTRAN Modules," Computer Sciences Corporation, Technical Memorandum, June 1984

⁵Card, D. N., F. E. McGarry, and G. T. Page, "Evaluating Software Engineering Technologies," *IEEE Transactions on Software Engineering*, July 1987

³Card, D. N., G. T. Page, and F. E. McGarry, "Criteria for Software Modularization," *Proceedings of the Eighth International Conference on Software Engineering*. New York: IEEE Computer Society Press, 1985

¹Chen, E., and M. V. Zelkowitz, "Use of Cluster Analysis To Evaluate Software Engineering Methodologies," *Proceedings of the Fifth International Conference on Software Engineering*. New York: IEEE Computer Society Press, 1981

⁴Church, V. E., D. N. Card, W. W. Agresti, and Q. L. Jordan, "An Approach for Assessing Software Prototypes," *ACM Software Engineering Notes*, July 1986

²Doerflinger, C. W., and V. R. Basili, "Monitoring Software Development Through Dynamic Variables," *Proceedings of the Seventh International Computer Software and Applications Conference*. New York: IEEE Computer Society Press, 1983

Doubleday, D., *ASAP: An Ada Static Source Code Analyzer Program*, University of Maryland, Technical Report TR-1895, August 1987 (NOTE: 100 pages long)

⁶Godfrey, S., and C. Brophy, "Experiences in the Implementation of a Large Ada Project," *Proceedings of the 1988 Washington Ada Symposium*, June 1988

⁵Jeffery, D. R., and V. Basili, *Characterizing Resource Data: A Model for Logical Association of Software Data*, University of Maryland, Technical Report TR-1848, May 1987

⁶Jeffery, D. R., and V. R. Basili, "Validating the TAME Resource Data Model," *Proceedings of the Tenth International Conference on Software Engineering*, April 1988

¹¹Li, N. R., and M. V. Zelkowitz, "An Information Model for Use in Software Management Estimation and Prediction," *Proceedings of the Second International Conference on Information Knowledge Management*, November 1993

⁵Mark, L., and H. D. Rombach, *A Meta Information Base for Software Engineering*, University of Maryland, Technical Report TR-1765, July 1987

⁶Mark, L., and H. D. Rombach, "Generating Customized Software Engineering Information Bases From Software Process and Product Specifications," *Proceedings of the 22nd Annual Hawaii International Conference on System Sciences*, January 1989

⁵McGarry, F. E., and W. W. Agresti, "Measuring Ada for Software Development in the Software Engineering Laboratory (SEL)," *Proceedings of the 21st Annual Hawaii International Conference on System Sciences*, January 1988

⁷McGarry, F., L. Esker, and K. Quimby, "Evolution of Ada Technology in a Production Software Environment," *Proceedings of the Sixth Washington Ada Symposium (WADAS)*, June 1989

McGarry, F., R. Pajerski, G. Page, et al., *Software Process Improvement in the NASA Software Engineering Laboratory*, Carnegie-Mellon University, Software Engineering Institute, Technical Report CMU/SEI-94-TR-22, ESC-TR-94-022, December 1994

³McGarry, F. E., J. Valett, and D. Hall, "Measuring the Impact of Computer Resource Quality on the Software Development Process and Product," *Proceedings of the Hawaiian International Conference on System Sciences*, January 1985

³Page, G., F. E. McGarry, and D. N. Card, "A Practical Experience With Independent Verification and Validation," *Proceedings of the Eighth International Computer Software and Applications Conference*, November 1984

¹²Porter, A. A., L. G. Votta, Jr., and V. R. Basili, *Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment*, University of Maryland, Technical Report TR-3327, July 1994

⁵Ramsey, C. L., and V. R. Basili, "An Evaluation of Expert Systems for Software Engineering Management," *IEEE Transactions on Software Engineering*, June 1989

- ³Ramsey, J., and V. R. Basili, "Analyzing the Test Process Using Structural Coverage," *Proceedings of the Eighth International Conference on Software Engineering*. New York: IEEE Computer Society Press, 1985
- ⁵Rombach, H. D., "A Controlled Experiment on the Impact of Software Structure on Maintainability," *IEEE Transactions on Software Engineering*, March 1987
- ⁸Rombach, H. D., "Design Measurement: Some Lessons Learned," *IEEE Software*, March 1990
- ⁹Rombach, H. D., "Software Reuse: A Key to the Maintenance Problem," *Butterworth Journal of Information and Software Technology*, January/February 1991
- ⁶Rombach, H. D., and V. R. Basili, "Quantitative Assessment of Maintenance: An Industrial Case Study," *Proceedings From the Conference on Software Maintenance*, September 1987
- ⁶Rombach, H. D., and L. Mark, "Software Process and Product Specifications: A Basis for Generating Customized SE Information Bases," *Proceedings of the 22nd Annual Hawaii International Conference on System Sciences*, January 1989
- ⁷Rombach, H. D., and B. T. Ulery, *Establishing a Measurement Based Maintenance Improvement Program: Lessons Learned in the SEL*, University of Maryland, Technical Report TR-2252, May 1989
- ¹⁰Rombach, H. D., B. T. Ulery, and J. D. Valett, "Toward Full Life Cycle Control: Adding Maintenance Measurement to the SEL," *Journal of Systems and Software*, May 1992
- ⁶Seidewitz, E., "Object-Oriented Programming in Smalltalk and Ada," *Proceedings of the 1987 Conference on Object-Oriented Programming Systems, Languages, and Applications*, October 1987
- ⁵Seidewitz, E., "General Object-Oriented Software Development: Background and Experience," *Proceedings of the 21st Hawaii International Conference on System Sciences*, January 1988
- ⁶Seidewitz, E., "General Object-Oriented Software Development with Ada: A Life Cycle Approach," *Proceedings of the CASE Technology Conference*, April 1988
- ⁹Seidewitz, E., "Object-Oriented Programming Through Type Extension in Ada 9X," *Ada Letters*, March/April 1991
- ¹⁰Seidewitz, E., "Object-Oriented Programming With Mixins in Ada," *Ada Letters*, March/April 1992
- ¹²Seidewitz, E., "Genericity versus Inheritance Reconsidered: Self-Reference Using Generics," *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications*, October 1994
- ⁴Seidewitz, E., and M. Stark, "Towards a General Object-Oriented Software Development Methodology," *Proceedings of the First International Symposium on Ada for the NASA Space Station*, June 1986

- ⁹Seidewitz, E., and M. Stark, "An Object-Oriented Approach to Parameterized Software in Ada," *Proceedings of the Eighth Washington Ada Symposium*, June 1991
- ⁸Stark, M., "On Designing Parametrized Systems Using Ada," *Proceedings of the Seventh Washington Ada Symposium*, June 1990
- ¹¹Stark, M., "Impacts of Object-Oriented Technologies: Seven Years of SEL Studies," *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications*, September 1993
- ⁷Stark, M. E. and E. W. Booth, "Using Ada to Maximize Verbatim Software Reuse," *Proceedings of TRI-Ada 1989*, October 1989
- ⁵Stark, M., and E. Seidewitz, "Towards a General Object-Oriented Ada Lifecycle," *Proceedings of the Joint Ada Conference*, March 1987
- ¹³Stark, M., and E. Seidewitz, "Generalized Support Software: Domain Analysis and Implementation," *Addendum to the Proceedings OOPSLA '94*, Ninth Annual Conference, Portland, Oregon, U.S.A., October 1994, pp. 8-13
- ¹⁰Straub, P. A., and M. V. Zelkowitz, "On the Nature of Bias and Defects in the Software Specification Process," *Proceedings of the Sixteenth International Computer Software and Applications Conference (COMPSAC 92)*, September 1992
- ⁸Straub, P. A., and M. V. Zelkowitz, "PUC: A Functional Specification Language for Ada," *Proceedings of the Tenth International Conference of the Chilean Computer Science Society*, July 1990
- ⁷Sunazuka, T., and V. R. Basili, *Integrating Automated Support for a Software Management Cycle Into the TAME System*, University of Maryland, Technical Report TR-2289, July 1989
- ¹³Thomas, W. M., A. Delis, and V. R. Basili, *An Analysis of Errors in a Reuse-Oriented Development Environment*, University of Maryland, Computer Science Technical Report, CS-TR-3424, UMIACS-TR-95-24, February 1995
- ¹⁰Tian, J., A. Porter, and M. V. Zelkowitz, "An Improved Classification Tree Analysis of High Cost Modules Based Upon an Axiomatic Definition of Complexity," *Proceedings of the Third IEEE International Symposium on Software Reliability Engineering (ISSRE 92)*, October 1992
- Turner, C., and G. Caron, *A Comparison of RAD and NASA/SEL Software Development Data*, Data and Analysis Center for Software, Special Publication, May 1981
- ¹⁰Valett, J. D., "Automated Support for Experience-Based Software Management," *Proceedings of the Second Irvine Software Symposium (ISS _92)*, March 1992
- ⁵Valett, J. D., and F. E. McGarry, "A Summary of Software Measurement Experiences in the Software Engineering Laboratory," *Proceedings of the 21st Annual Hawaii International Conference on System Sciences*, January 1988

¹⁴Waligora, S., J. Bailey, and Mike Stark, "The Impact of Ada and Object-Oriented Design in NASA Goddard's Flight Dynamics Division," March 1995

³Weiss, D. M., and V. R. Basili, "Evaluating Software Development by Analysis of Changes: Some Data From the Software Engineering Laboratory," *IEEE Transactions on Software Engineering*, February 1985

⁵Wu, L., V. R. Basili, and K. Reed, "A Structure Coverage Tool for Ada Software Systems," *Proceedings of the Joint Ada Conference*, March 1987

¹Zelkowitz, M. V., "Resource Estimation for Medium-Scale Software Projects," *Proceedings of the Twelfth Conference on the Interface of Statistics and Computer Science*. New York: IEEE Computer Society Press, 1979

²Zelkowitz, M. V., "Data Collection and Evaluation for Experimental Computer Science Research," *Empirical Foundations for Computer and Information Science* (Proceedings), November 1982

⁶Zelkowitz, M. V., "The Effectiveness of Software Prototyping: A Case Study," *Proceedings of the 26th Annual Technical Symposium of the Washington, D.C., Chapter of the ACM*, June 1987

⁶Zelkowitz, M. V., "Resource Utilization During Software Development," *Journal of Systems and Software*, 1988

⁸Zelkowitz, M. V., "Evolution Towards Specifications Environment: Experiences With Syntax Editors," *Information and Software Technology*, April 1990

¹⁴Zelkowitz, M. V., "Software Engineering Technology Infusion Within NASA," *IEEE Transactions On Engineering Management*, vol. 43, no. 3, August 1996

NOTES:

¹This article also appears in SEL-82-004, *Collected Software Engineering Papers: Volume I*, July 1982.

²This article also appears in SEL-83-003, *Collected Software Engineering Papers: Volume II*, November 1983.

³This article also appears in SEL-85-003, *Collected Software Engineering Papers: Volume III*, November 1985.

⁴This article also appears in SEL-86-004, *Collected Software Engineering Papers: Volume IV*, November 1986.

⁵This article also appears in SEL-87-009, *Collected Software Engineering Papers: Volume V*, November 1987.

⁶This article also appears in SEL-88-002, *Collected Software Engineering Papers: Volume VI*, November 1988.

⁷This article also appears in SEL-89-006, *Collected Software Engineering Papers: Volume VII*, November 1989.

⁸This article also appears in SEL-90-005, *Collected Software Engineering Papers: Volume VIII*, November 1990.

⁹This article also appears in SEL-91-005, *Collected Software Engineering Papers: Volume IX*, November 1991.

¹⁰This article also appears in SEL-92-003, *Collected Software Engineering Papers: Volume X*, November 1992.

¹¹This article also appears in SEL-93-001, *Collected Software Engineering Papers: Volume XI*, November 1993.

¹²This article also appears in SEL-94-004, *Collected Software Engineering Papers: Volume XII*, November 1994.

¹³This article also appears in SEL-95-003, *Collected Software Engineering Papers: Volume XIII*, November 1995.

¹⁴This article also appears in SEL-96-001, *Collected Software Engineering Papers: Volume XIV*, October 1996.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE October 1996		3. REPORT TYPE AND DATES COVERED Technical Memorandum
4. TITLE AND SUBTITLE Software Engineering Laboratory Series Collected Software Engineering Papers: Volume XIV			5. FUNDING NUMBERS Code 551	
6. AUTHOR(S) Flight Dynamics Systems Branch				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS (ES) Goddard Space Flight Center Greenbelt, Maryland 20771			8. PERFORMING ORGANIZATION REPORT NUMBER SEL-96-001	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS (ES) National Aeronautics and Space Administration Washington, DC 20546-0001			10. SPONSORING / MONITORING AGENCY REPORT NUMBER TM—1998—208613	
11. SUPPLEMENTARY NOTES				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Unclassified-Unlimited Subject Category: 82 61 Report available from the NASA Center for AeroSpace Information, 7121 Standard Drive, Hanover, MD 21076-1320. (301) 621-0390.			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) The Software Engineering Laboratory (SEL) is an organization sponsored by NASA/GSFC and created to investigate the effectiveness of software engineering technologies when applied to the development of application software. The activities, findings, and recommendations of the SEL are recorded in the Software Engineering Laboratory Series, a continuing series of reports that includes this document.				
14. SUBJECT TERMS Software Engineering Laboratory, Application software, Documentation			15. NUMBER OF PAGES 206	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	